Мирослав Д. Лутовац

Интернет програмирање

Висока школа електротехнике и рачунарства
струковних студија, Београд

Академија техничко-уметничких струковних студија
Београд

Београд 2020

Наслов: Интернет програмирање

Аутор: др Мирослав Д. Лутовац

Прво издање

Рецензент: др Перица Штрбац
др Владимир Младеновић

Издавачи: Висока школа електротехнике и рачунарства струковних студија, Београд
Академија техничко-уметничких струковних студија Београд

За издавача: др Вера Петровић

Дизајн насловне стране: Ненад Толић

Штампа: Развојно-истраживачки центар графичког инжењерства ТМФ, Београд

Тираж: 30

ISBN: 978-86-7982-324-3

CIP - Каталогизација у публикацији - Народна библиотека Србије, Београд
004.738.5:004.42(075.8)
004.438JAVA(075.8)
ЛУТОВАЦ, Мирослав, 1957-
Интернет програмирање / Мирослав Д. Лутовац. - 1. изд. - Београд : Висока школа електротехнике и рачунарства струковних студија : Академија техничко-уметничких струковних студија, 2020 (Београд : Развојно-истраживачки центар графичког инжењерства ТМФ). - 274 стр. : илустр. ; 24 cm
Тираж 30. - Библиографија: стр. 265. - Регистар.
ISBN 978-86-7982-324-3 (ВШЕРСС)
a) Интернет - Програмирање b) Програмски језик "Java"

COBISS.SR-ID 21721865

САДРЖАЈ

Предговор

Књига је написан на основу материје коју је излагао аутор у периоду од 2017. до 2020. студентима Високе школе електротехнике и рачунарства струковних студија у Београд који су предмет “Интернет програмирање” слушали као изборни Циљ предмета је оспособљавање студената да пројектују и пишу савремене Интернет апликације коришћењем Јава програмског језика.

“Интернет програмирање” спада у групу предмета која се под различитим називима може срести у студијским програмима Софтверско инжењерство. Софтверско инжењерство је област која пружа студентима могућност брзог проналажења посла и добре зараде, па тако привлачи све већи број заинтересованих студената, и све веће могућности за реализацију пројеката у скоро свим сегментима живота.

Овај уџбеник је делом објављен и под насловом Јаваскрипт и сервлети, програмирање интернет апликација, аутора-издавача др Мирослав Д. Лутовац, и штампано у папирном облику код Line Print, Београд, ISBN: 978-86-88443-03-6. Након потписивања ауторског уговора сва права су пренета са Мирослава Лутовца на Одсек Висока школа електротехике и рачунарства Академије техничко-уметничких струковних студија Београд.

15. септембар 2020. У Београду

Др Мирослав Д. Лутовац, дипл. инж.,

професор Високе школе електротехнике и рачунарства струковних студија у Београду,

1 Основни појмови

Резиме

Циљ овог поглавља је да студент научи и постане фамилијаран са основном појмовима за које се подразумева да их зна пре него што почне да пише програме за потребе Интернет програмирања. Како је у школи већина предмета изборна, неки студенти неће имати детаљно знање из сродне материје. Због тога се у овом поглављу излаже онај део градива који је неопходан за успешно програмирање интернет апликација.

Увод у основне појмове

За успешно савладавање градива, које се изучава у оквиру предмета Интернет програмирање, неопходно је разумети окружење у оквиру кога се ови програми користе и јасан циљ шта се жели постићи написаним програмима. Градиво из овог предмета се може наћи у бројним књигама које на први поглед имају различите називе, што може довести до погрешног интерпретирања суштине предмета. На пример Бошко Николић, професор Електротехничког факултета у Београду, који материју из ове области предаје или је предавао на већем броју високошколских институција, има објављен уџбеник Програмирање интернет апликација, што јасно указује да се у оквиру овог предмета изучава програмирање апликација које се користе на Интернету [2017_Nikolic_PIAuszz], [2008_Nikolic_IP]. То значи да се из овог предмета не програмира сам Интернет као скуп повезаних рачунарских мрежа, већ се програмирају апликације које се праве са одређеном наменом, на пример за пословне системе. Често се користе и термини веб програмирање или програмирање мреже, где се под мрежом подразумева Интернет. Иако се под појмом мреже подразумева систем повезаних рачунара, а појам веб је сервис који се користи на мрежама, у оба случаја се подразумева да се генеришу и користе програми написани за апликације које имају корист за оне који су повезани у мрежу или користе одговарајуће сервисе. Апликације које се називају веб прегледачи (*Web browsers*) омогућавају да се једноставно приступи вебу и да се подаци размењују између великог броја рачунара повезаних на Интернет. Неке од познатијих апликација су веб-меил, сајтови за продају, онлајн аукције, сервиси за слање порука, вики портали. Уобичајени енглески називи за овакву апликацију је *web application* или *web app*, односно у множини *web applications* или *web apps*. У већини случајева се ради о појединачним захтевима да се добију или размењују подаци, а оне који шаљу захтеве називамо клијентима (*client*). Једном клијенту

или већем броју клијената треба одговорити на ове појединачне захтеве, и страну којој се обраћа клијентима називамо сервер (*server*). Стога се оваква врста апликација у којој клијенти постављају захтеве, а са друге стране је сервер који одговара на захтеве свих клијената назива клијент-сервер рачунарски програм (*client–server computer program*). На страни клијента је прегледач који прима поруке у текстуалном формату и на основу јасно утврђених правила приказује клијенту одговоре сервера. Са стране сервера, примљени захтеви се анализирају, обрађују и у складу са написаним програмима сервер генерише текстуалне поруке које се шаље клијентима, с тим да се води рачуна о томе да прегледач на страни клијента може исправно да прикаже одговор, односно да клијентски рачунар има програме који ће правилно интерпретирати одговоре добијене од сервера.

На основу овога се може разумети шта су циљ и сврха овог предмета: како написати захтев који клијент преко прегледача шаље серверу, и како програм на страни сервера треба да интерпретира захтев, како да га обради и како да клијенту пошаље поруку са одговором који клијентов прегледач може да разуме и прикаже на страни клијента.

У оквиру овог предмета се подразумева да је читаоцима позната структура текстуалних порука и протокол којим се поруке преносе. У циљу ефикасније обуке, неки од појмова који су важни за разумевање израде интернет апликација биће објашњени у најосновнијим карактеристикама.

1.1 Рачунарска мрежа

Резиме

Циљ овог поглавља је да се студент упозна са појмовима о инфраструктури на којој се заснива интернет програмирање и да разуме архитектуру рачунарске мреже преко које се остварује веза између клијента и сервера и користе веб апликације. Дати су најосновнији појмови за повезивање рачунара (класично, P2P, PPP, преко PSTN), клијент-сервер, TCP/IP и OSI модел, IP протокол, мрежни уређаји (хаб, рутер, свич, бежична приступна тачка, блутут) и IP адреса.

Увод у рачунарске мреже

Сваки рачунар може да ради независно од рада свих осталих рачунара и не мора да буде повезан ни са једним другим рачунаром. Међутим, када је потребно да се размене подаци са другим рачунаром или са више других рачунара, неопходно је обезбедити везу преко које се остварује пренос података. Иако се уобичајено мисли да се та веза остварује преко проводних жица, у данашње време се све више користе различити путеви преноса, па и различити уређаји, тако да са становишта овог предмета нас не интересује како се конкретно и физички преносе подаци. Једино што нас интересује јесте да један рачунар шаље поруку неком другом рачунару. Подаци који се шаљу не морају увек да стигну до свог одредишног уређаја, иако се подразумева да ће за коначно време послата порука бити испоручена.

За пренос сваке поруке је потребно време док је одредишни рачунар не прими и детектује поруку, и све се одвија са кашњењем у односу на тренутак слања. Сматраћемо да се пренос порука одвија у разумном периоду и тада кажемо да се комуникација одвија у

реалном времену. Сам пут преноса може да буде преко проводних жица, оптичког кабла, електромагнетних таласа, и порука може мењати свој облик од текста у бројеве, модулисане сигнале, а на крају се на одредишту поново претвара у оригинални текст или поруку представљену бројевима. Како се користе два рачунара као крајње тачке које размењују поруке, један рачунар на месту слања поруке и један рачунар на месту пријема те поруке, подразумева се да је порука дигитална (коначан скуп бројева у одређеним дискретним временски тренуцима). Да би се избегла анализа какав је то уређај који шаље поруке, а какав онај који прима поруке, и какве су све трансформације сигнала биле коришћене при преносу података, посматраћемо рачунар који шаље или прима поруку као чвор мреже (*node*). У пракси постоји велики број уређаја који се такође називају чворовима, као што су штампачи (*printers*), рутери (*routers*), и други (*switches, bridges, gateways, terminals*), али у оквиру веб апликација нас интересују само чворови који су рачунари у којима су прегледачи и програми за рад са овим апликацијама. Често се користи и термин *host* у ситуацијама када је чвор рачунар опште намене (*general-purpose computer*). На месту клијента уместо класичног рачунара може бити мобилни телефон, таблет или лаптоп рачунар, па и рачунари величине кредитне картице.

1.1.1 Повезивање рачунара

Да би два рачунара размењивала податке један са другим, односно да би се остварила комуникација између два рачунара, неопходно је да су повезани. Један од начина да се два рачунара повежу јесте да рачунари имају прикључак (конектор) који се назива порт. Кабл који има одговарајуће адаптере на својим крајевима може да се повеже на конекторе рачунара. Један од некада најраспрострањенијих начина повезивања су серијски и паралелни прикључак, који су коришћени да би се рачунар повезао са штампачем. Неки од најпознатијих прикључака су *USB* прикључак (*Universal Serial Bus*), серијски прикључак (на пример *RS*-232), паралелни прикључак (користили су се и називи принтер порт, штампач порт), прикључак за монитор, компактни прикључак за видео и звучне записе (*HDMI*, *High-Definition Multimedia Interface*), мини *DIN* прикључак (*Deutsches Institut für Normung*), *DVI* прикључак (*Digital Visual Interface*), *SCSI* прикључак (*Small Computer System Interface*), *PS*/2 прикључак (*IBM Personal System*/2), *Ethernet* прикључак. За повезивање два рачунара преко *USB* портова може да буде потребан посебан кабла опремљен уређајем *USB Bridge* (користе се и термини *USB Bridge Cable, USB Link Cable, USB Networking Cable*).

За повезивање рачунара најчешће се подразумева да је веза жична или оптичка. Уз одговарајућу опрему или хардверске додатке може се користити и бежична метода повезивања као што је *Wi-Fi* и *Bluetooth* стандард.

Вај-фај (*Wi-Fi, Wireless-Fidelity*) је бежична локална рачунарска мрежа која почива на стандарду IEEE 802.11.

802.11*b* је имао пропусност од 11Mbps уз *fallback rate* на 5.5, 2 и 1 MB/s, тако да ако је интензитет сигнала мањи, или је удаљеност између уређаја већа, мрежна опрема се прилагођава условима и смањује брзину везе. Већина *WLAN* и *WAN* мрежа ради на 802.11*b* стандарду па је постао референца за *WLAN*. Примарна сигурносна енкрипција је *WEP* (*Wired Equivalent Privacy*) у 64, 128 или 256 битном облику.

802.11*g* је верзија стандарда која може достићи пропусност до 54 Mb/s. Ови уређаји имају паметније *autofallback rate*, па брзина са удаљеношћу може значајно да опадне.

802.11*n* је новија верзија стандарда која достиже пропусност до 600 Mb/s, и користи *MIMO* (*multiple-input multiple-output*) технологију за постизање већих брзина од 802.11*b* и 802.11*g* стандарда. Има побољшања у врстама заштите. Уређаји који раде у овом стандарду обично користе две или више антена. Могу да раде и у 802.11*b* и 802.11*g* стандардима уколико други уређаји у мрежи не дозвољавају брзине и могућности *n* стандарда. Неки *Wi-Fi* штампачи не могу да раде у стандарду 802.11*n*, и подржавају само стандарде 802.11*b* и 802.11*g*, због чега се појављују проблеми у инсталирању мрежног бежичног штампача, као што је на пример *HP LaserJet Pro P1102w Printer*.

Савремени рачунари немају неке од наведених врста конектора, као ни све могућности повезивања, а хардверска надоградња за неке врсте повезивања је већ укључена у интегрисана кола која се користе, па више нису потребни додаци и мрежне картице. Најчешће оба рачунара имају инсталиран и програм који ће управљати разменом података између рачунара. Док је у ранијим верзијама доминантна сврха комуникације била између два рачунара, сада се очекује да велики број рачунара буде међусобно повезан. Постојање стандардизованих протокола, који се користе на Интернету, били су пресудни да се савремени рачунарски уређаји повезују на јединствен стандардизован начин.

Подразумева се да уређаји засновани на рачунарима имају могућност повезивања преко Интернета, па ће и сва даља материја у оквиру овог предмета бити усмерена на овај вид комуникације; рачунари су повезани на светску рачунарску мрежу Интернет и користе стандарде за комуникацију који су познати и расположиви.

1.1.2 Повезивање рачунара *P2P*

Ако се размена одвија само између два рачунара, може да се користи модел комуникације *P2P* односно *peer to peer*, који се најчешће користи за дељење датотека. *P2P* је скраћеница од енглеског израза *peer to peer*, што говори да се размена података обавља између два равноправна учесника, или једнак једнаком, или жаргонски вршњак вршњаку. Заједничко коришћење фајлова који се налазе на оба рачунара значи да са оба рачунара може да се приступи овим фајловима, и користи се енглески термин *share* фајлова. Типична примена је да више рачунара има конекцију *P2P* са штампачем, па је на овај начин штампач заједнички ресурс за све рачунаре повезане са штампачем. Овај протокол је прихваћен и имплементиран у Мајкрософтовим оперативним системима. Под термином протокол подразумева се скуп правила и конвенција за размену информација у оваквој врсти комуникације.

Скајп (*Skype*) је пример *peer-to-peer* (*P2P*) мреже (и назив истоименог програма) коју је направила компанија *Skype Technologies*, а коју је купио Мајкрософт и пружа услугу интернет телефоније (ВоИП, енглески *Voice over Internet Protocol*, *VoIP*). Корисници програма могу бесплатно међусобно да причају преко Интернета или локалне мреже, а програм пружа и услугу позивања и примања позива са стандардних телефонских бројева (*SkypeOut* и *SkypeIn*) која се додатно наплаћује.

P2P може да се користи и за размену података између корисника и путем Интернета, али и без повезивање на Интернет. Коришћењем *P2P* избегава се повезивање рачунара преко

сервера. Свака два рачунара која су повезана могу да преузимају, преносе и размењују фајлове.

Уобичајено је да се *P2P* мрежа користи за повезивање до 10 рачунара. Због удобности рада, сви повезани рачунари имају инсталиране идентичне одговарајуће софтвере, на пример за обраду текста.

1.1.3 Повезивање рачунара *PPP*

За остваривање директне везе између два чвора рачунарске мреже користи се *Point-to-Point Protocol* (*PPP*). За повезивање рачунара се најчешћи користи проводни кабл, телефонске линије, оптичка влакна или *UTP* мрежни каблови. Већина интернет компанија користе *PPP* за *dial-up* приступ Интернету. Такође је могућ *PPP* преко Етернета (*Ethernet, PPPoE*), повезујући неки *DSL* модем са рачунаром преко мрежне картице што је много брже него повезивање са *USB* или неком другом сабирницом. *PPP* је дизајниран да ради са протоколима трећег слоја *OSI* референтног модела.

1.1.3.1 *DSL* модем

Дигитална претплатничка линија, за коју се користи термин *Digital Subscriber Line* (*DSL*) модем, је уређај који се користи да повеже два рачунара или рачунар и рутер преко телефонске линије при чему се преко дигиталне телефонске централе остварује дигитална претплатничка линија. Модем је уређај који остварује дуплекс везу, односно преко њега се истовремено и примају и шаљу подаци.

У Србији се највише користи асиметрична дигитална претплатничка линија (A*DSL* - *Asymetric Digital Subscriber Line*). Карактеристика A*DSL* технологије је већа брзина преноса података са Интернета од брзине слања података ка Интернету, што одговара реалним потребама корисника који обично много више података примају (*download*) него што их шаљу (*upload*).

1.1.4 Повезивање рачунара преко *PSTN*

Развој телекомуникација био је заснован на идеји да је за сваки сервис грађена потребна посебна инфраструктура. Јавна мрежа је коришћена за телефонски сервис, телеграфски сервис и сервис за пренос података. Веза између телефонског претплатника и телефонске централе највећим делом је ишла преко бакарних парица, такозваних телефонских линија (бакарних водова-парица). Да би два корисника остварила везу, они су физички морали да заузму цео пут преко телефонских линија, а телефонске централе су служиле за преспајање, за које се користи и термин комутација канала. Енглески назив који се користи је *PSTN* (*Public Switched Telephone Network*) и који означава да се ради о интегрисаном систему који примењује технологију комутације канала; суштина је да се свакој говорној вези додељује физички канал између два претплатника за сваки појединачни разговор. Овакве интегрисане мреже још увек постоје на локалном, националном и интернационалном нивоу и представљају основну инфраструктуру за јавне телекомуникације. Данас се *PSTN* мреже састоје од бакарних телефонских линија, радио релејних система преноса, оптичких каблова, сателитских линкова и мобилних мрежа. Осим фиксних аналогних линија до крајњих корисника, *PSTN* систем омогућава и пренос дигиталних сигнала. Због високих трошкова инфраструктуре, везе између телефонских централа се остварује преко мање паралелних канала. На пример, за хиљаду

претплатничких телефонских прикључака на телефонски централу, за комуникацију до следеће телефонске централе користи се 100 канала. Због малог трајања говорног телефонског заузећа, не би било исплативо да се заузима више од 100 канала. Међутим, повезивањем претплатника рачунарима и коришћењем модема, трајање заузете везе постаје значајно веће, па је за дуготрајне везе недовољно само 100 канала између централа.

Све веће потребе корисника за новим сервисима као што су видео на захтев, рад од куће, учење на даљину, мрежна куповина, интерактивне мрежне игре, радио и ТВ, поставиле су нове захтеве за коришћења постојеће инфраструктуре и изградњу нових инфраструктура, са кључним карактеристикама да се великом броју корисника обезбеди истовремено коришћење инфраструктуре, а да то не захтева значајније финансијске издатке за претплатнике. Свакако, претплатник који жели да има боље карактеристике за комуникацију и размену података мора и да плати те погодности.

1.1.5 Клијент- сервер модел

Заузимање канала од стране два рачунара, по коме би постојала комуникација само за њих, економски би била неисплатива због високих трошкова инфраструктуре. Свакако, када би претплатници прихватили да плате овај износ, онда би била остварена таква комуникација. У оваквим случајевима би корисник плаћао закуп претплатничке линије коју би само он користио, односно то би била изнајмљена претплатничка линија.

Савремени системи су у све већој мери мобилни, па фиксна места конекције постају непрактична. Већина система очекује размену или пружање података већем броју корисника, па би били потребни посебни уређаји за мултиплексирање са више линија на један порт. Због тога инфраструктура која је по моделу комуникације *P2P* (*peer to peer*) и *PPP* (*Point-to-Point Protocol*) није практична за ове апликације ако се не користи Интернет. Постојећа интернет мрежа може да се искористи за клијент-сервер модел који повезује два или више рачунара преко Интернета (или рачунарске мреже која користи интернет протоколе). Најчешће већи број рачунара које називамо клијенти и један сервер комуницирају преко рачунарске мреже на одвојеним хардверима, али један рачунар може истовремено да буде и клијент и сервер. За тестирање рада интернет апликација у овом курсу ми ћемо конфигурисати да један рачунар размењује податке између клијента и сервера који су на истом рачунару. Када је апликација тестирана, серверски део софтвера може бити пребачен на други рачунар и тада се користи мрежа да преко ње комуницирају клијентски рачунари са сервером.

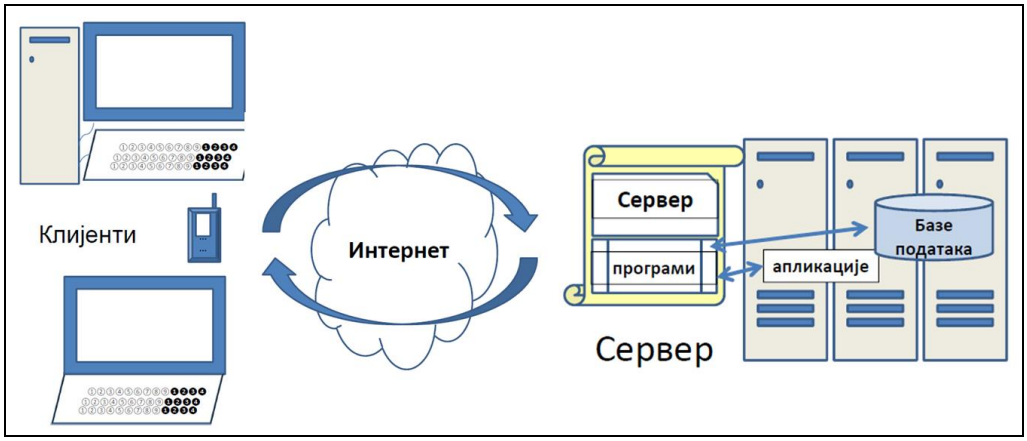
Клијент-сервер модел је структура која у принципу подразумева да су рачунари удаљени; рачунари које називамо клијентима су они који захтевају услуге од сервера; рачунари који обезбеђују ресурсе и услуге на основу примљених захтева од клијената, називамо сервери. Сервер покреће серверске програме након пријема захтева од клијентских рачунара, и он је нека врста домаћина који опслужује клијенте због чега се назива и *host*, односно на серверу се покрећу програми који деле своје ресурсе са клијентима. Клијент од сервера захтева серверов садржај или услугу и не дели своје ресурсе. Клијенти започињу сесију комуникације са серверима који чекају долазеће захтеве. Примери компјутерских апликација који користе клијент-сервер модел су имејл и веб сервери који омогућава преглед веб страница.

Да би комуницирали користећи клијент-сервер модел, рачунари морају да користе исти језик и да раде по унапред утврђеним правилима. Правила комуникације и језик називају се протоколи за комуникацију. Клијент-сервер протоколи функционишу у слоју апликација. Сервер може да имплементира апликациони програмски интерфејс (АПИ, (АПИ, енглески *Application programming interface, API*). АПИ је слој апстракције за приступање услузи, што олакшава размену података на различитим платформама.

Треба уочити двозначност речи хост (*host*). Уз појам хост може се везати реч клијент или корисник и тада се подразумева да је то било који рачунар повезан са мрежом. Док се речи сервер и клијент могу односити и на рачунар и на рачунарски програм, *server-host* и *user-host* увек се односе на рачунаре. Хост (домаћин) је вишенаменски рачунар, а клијенти и сервери су само програми који се покрећу на хосту. У моделу клијент-сервер, клијент прослеђује захтеве на које сервер шаље одговоре.

Како је већи број клијентских рачунара повезан са серверским рачунаром, можемо да посматрамо да је сервер центар догађања, и да се у ширем смислу овакав систем посматра као централизовано рачунарство. Сам серверски рачунар може бити повезан и са другим рачунарима са којих преузима садржаје или их користи да би урадио одређени посао, а да затим те садржаје проследи корисничким рачунарима. Циљ је да се што је могуће више складиштених података и обраде захтева пребаци са клијентског рачунара на сервер, а да за потребе сервера те задатке ураде други рачунари. То омогућава да комуникација између клијента и сервера буде једноставнија и довољно брза.

Некада су клијенти били само рачунарски терминал који немају оперативни систем и служили су само као улазно/излазни интерфејс за комуникацију са сервером. Садашњи рачунари имају на располагању ресурсе и да складиште и обрађују податке, и не би морали да се ослањају на сервер. Међутим, примену апликација које ћемо ми разматрати у овом курсу подразумевају могућност да већи број удаљених рачунара има приступ заједничким подацима који се динамички мењају, па подаци који се налазе код клијента не могу увек да буду ажурни и тачни. На пример, ако се ради о серверу који се користи за продавницу књига, сервер има приступ бази података са тренутним стањем књига на лагеру; када један клијент преко Интернета купи књигу, информација да је на лагеру мање књига се одмах ажурира у електронској продавници; када неки други клијент жели да купи књигу, он види да је смањен број књига на лагеру за 1. Овакве потребе да сви клијенти имају ажурне податке преко серверског рачунара су довеле до сервисно орјентисаних архитектура, а то је убрзало развој рачунарства у облаку.



Слика 1.1.1 Клијент- сервер модел.

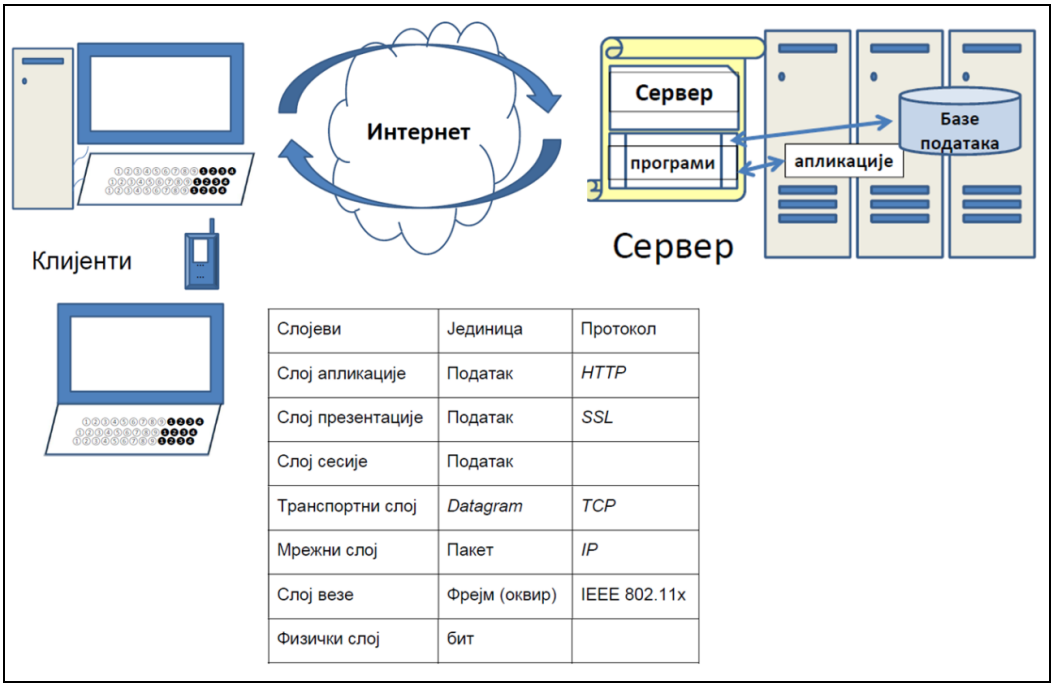
У овом курсу циљ је да се студенти оспособе да разумеју а затим и напишу програме који ће омогућити да се на страни клијентског рачунара прикаже садржај за који су клијенти заинтересовани, да попуне електронске формуларе и пошаљу захтев серверском рачунару, да на страни клијента провере да ли су све исправно урадили и коригују своје захтеве и податке које треба да пошаљу серверу, а да затим напишу програме на страни сервера који ће анализирати клијентске захтеве, урадити потребну обраду података, прикупити потребне податке са других рачунара сервисно орјентисаној архитектури и пошаљу одговор клијентском рачунару на захтев који су примили.

Слика 1.1.1 приказује клијент- сервер модел где се као примери клијената појављују један класичан рачунар, један мобилни телефон и један лаптоп. На страни сервера је неколико рачунара који функционишу као систем, на коме се налазе програми за комуникацију са клијентима, апликације које извршавају специфичне захтеве клијената или обрађују податке, као и рачунари на којима су базе података.

Комуникациони медијум преко кога клијенти приступају серверу је Интернет, као рачунарска мрежа, а на који је такође повезан и сервер. У овом тренутку нас не интересује како су клијентски и серверски рачунари повезани на Интернет, и да ли је то кабловска или бежична конекција. Број клијената може бити значајно већи, а преко Интернета се може доћи и до већег броја сервера. Како клијенти проналазе сервере, и који рачунарски језик користе, биће објашњено у следећим поглављима.

1.1.6 *TCP/IP* и *OSI* модел

Да би подаци били послати са једног рачунара на други, они морају да буду у таквом облику да задовољавају одређена правила. У рачунарској терминологији се користи ОСИ референтни модел или референтни модел за отворено повезивање система (*OSI, Open Systems Interconnection Basic Reference Model*) . ОСИ модел се користи да се објасни како се подаци преносе и обрађују у хардверу и софтверу, као што то илуструје Слика 1.1.2. ОСИ модел се састоји од седам логичких нивоа који се називају и слојеви.



Слојеви	Јединица	Протокол
Слој апликације	Податак	*HTTP*
Слој презентације	Податак	*SSL*
Слој сесије	Податак	
Транспортни слој	*Datagram*	*TCP*
Мрежни слој	Пакет	*IP*
Слој везе	Фрејм (оквир)	IEEE 802.11x
Физички слој	бит	

Слика 1.1.2 Клијент-сервер модел са ОСИ слојевима и протоколима.

Табела 1.1 показује слојеве ОСИ модела и протоколе *TCP/IP* и како се и у ком слоју ОСИ модела за пренос користе протоколи *IP* и *TCP*. У сваком слоју се обавља трансформација из једног облика сигнала у други, или се подаци пакују у одређену форму, што се често назива енкапсулација.

Физички слој (*Physical Layer*) је најнижи слој у ОСИ моделу рачунарске мреже који има функцију да електричне сигнале хардверских компоненти претвори у облик који је погодан да се пошаље преко слоја везе, и наравно да сигнал који долази из слоја везе конвертује у облик који ће бити прослеђен хардверским компонентама.

Слој везе (*Data Link Layer*) је слој у ОСИ моделу у коме се праве пакети који ће бити пренети између два чвора. У овом слоју се могу користити различити протоколи који користе различите методе енкапсулације пакета у фрејмове за пренос путем различитих медија.

Табела 1.1.1 Слојеви ОСИ модела и *TCP/IP*.

<table>
<tr><th>ОСИ модел</th><th>TCP/IP</th></tr>
<tr><td>Слој апликације
Мрежни процеси везани за апликацију</td><td rowspan="3">Слој апликације</td></tr>
<tr><td>Слој презентације
Енкрипција и кодирање података</td></tr>
<tr><td>Слој сесије
Успостављање сесије крајњих корисника</td></tr>
<tr><td>Транспортни слој
Веза и транспорт</td><td>Транспортни слој, TCP</td></tr>
<tr><td>Мрежни слој
Логичко адресирање и рутирање</td><td>Интернет слој, IP</td></tr>
<tr><td>Слој везе
Физичко адресирање, приступ медијуму</td><td rowspan="2">Слој приступа мрежи</td></tr>
<tr><td>Физички слој
Трансмисија сигнала</td></tr>
</table>

Слој мреже (*Network Layer*) је слој у ОСИ моделу који користи протоколе и сервисе и који обезбеђују идентификацију крајњих корисника мреже, као и путање између њих. Да би пакети којима се преносе подаци стигли од једног чвора мреже до другог, потребно је да пакети садрже адресу коме се шаље, али и ко шаље, како би на месту пријема могло да се идентификује ко је послао податке. Слој мреже поред других протокола садржи *IP* протокол.

Транспортни слој (*Transport Layer*) управља пакетима који путују између два рачунара. Неке од функција овог слоја су праћење размене података, спајање сегмената порука на нижим нивоима, и идентификација апликације. Транспортни слој поред других протокола садржи и *TCP* протокол.

Слој сесије (*Session Layer*) обавља успостављање везе између крајњих рачунара и синхронизује њихову размену података, успоставља и одржава сесије између покренутих програма на предајној и пријемној страни.

Слој презентације (*Presentation Layer*) у ОСИ моделу намењен је да одговара на захтеве слоја апликације и даље их прослеђује слоју сесије. Главне функције су кодирање и конверзија података, компресија и декомпресија података при слању и пријему података, енкрипција и декрипција података.

Слој апликације (*Application Layer*) у ОСИ моделу има улогу да обави размену података између апликација у мрежи и одговарајућих сервиса и протокола. На овом слоју програмер користи *API* за остваривање мрежне комуникације.

Претпоставимо да сервер треба да пошаље податке клијенту. Сервер је предајни уређај, а клијентов рачунар је пријемни рачунар. У слоју апликације се ови подаци припремају као пакет и додаје се адреса ко треба да прими пакет, слично као да у локалној пошти где станујете адресирате коверту у којој је садржај који шаљете некоме. Додата адреса на коверти у којој је садржај писма (где су подаци који се шаљу), заједно са садржајем, чини целину. Додата адреса се назива заглавље (на енглеском *header*). Овакав пакет се прослеђује следећем слоју, где се додаје ново заглавље. На пример, од локалне поште пошиљка се прослеђује главној отпремној пошти, ту се разврстава у збирну пошиљку која се шаље у неку европску пошту. Сваки пут када се пошиљка шаље следећем слоју, додаје се заглавље како би они који прослеђују пошиљку знали којим путем и превозним средствима пошиљка треба да се креће. Када пошиљка дође до транспортног слоја, додаје се информација према *TCP* протоколу. Информације о *TCP* су сада у заглављу пакета. У следећем слоју се додају адресе уређаја који се користе за пренос пошиљке. Када се дође до физичког слоја, подаци се претварају у низ јединица и нула и шаљу кроз одговарајући медијум. Због могућности погрешно пренесених јединица и нула, додају се бити за контролу исправности преноса; примењују се разноврсни алгоритми. На уређају који прима низ послатих 1 и 0, прво се контролише да ли су сви пренети бити исправни, и ако се детектује грешка, покушава се проналажење грешке и корекција погрешно примљених бита. Затим се од физичког слоја подаци преносе ка апликативном слоју на основу информација из заглавља пакета. Када подаци стигну до пријемника на месту клијентског рачунара, ти подаци се обрађују и користе у складу са садржајем послате серверске поруке.

Табела 1.1.2 приказује врсту података који се преносе преко мреже у одговарајућим слојевима и протоколе који су интересантни за оне који се баве програмирањем интернет апликација.

Слика 1.1.2 илуструје размену података између клијената и сервера где се сада види шта се дешава са подацима које клијент шаље ка серверу, и подаци које сервер шаље ка клијенту. Са становишта програмирања интернет апликација, програмер не мора да води рачуна о преносу података преко Интернета, већ само о садржају који треба да се пренесе.

Табела 1.1.2 Врсте података и неки коришћени протоколи

Слојеви	Јединица	Протокол
Слој апликације	Податак	*HTTP*
Слој презентације	Податак	*SSL*
Слој сесије	Податак	
Транспортни слој	*Datagram*	*TCP*
Мрежни слој	Пакет	*IP*
Слој везе	Фрејм (оквир)	IEEE 802.11x
Физички слој	бит	

Додатна објашњења у вези *TCP* и *IP* протокола су неопходна да би се разумела улога у интернет апликацијама.

1.1.7 *IP* протокол

Интернет протокол (*Internet Protocol, IP*) је протокол који садржи информације о адресирању, тако да сви мрежни уређаји (рачунар, сервер, интерфејс рутера) који су повезани на Интернет имају своју јединствену адресу и тиме се идентификују у целој интернет мрежи. *IP* садрже и контролне информације које омогућују пакетима да буду прослеђени (рутирани) на основу познатих *IP* адреса. Протокол је документован стандардом *RFC* 791 и представља са *TCP* протоколом језгро интернет протокола.

IP не захтева претходно успостављање везе у тренутку слања податка, већ рачунар шаље податке у пакетима све док не проследи поруку. Пренос података је непоуздан и нема гаранције да ће послати пакет доћи до одредишног рачунара. Пакет у процесу преноса може да се промени, у зависности којим путем се шаље, а може се догодити да делови пакета као сегменти не стигну до одредишта по редоследу слања, може стићи више копија послатих сегмената, али и део података може остати неиспоручен. Да би се повећала поузданост испоруке садржаја, *IP* се користи у комбинацији са *TCP* протоколом који се налази у слоју изнад *IP* протокола. *TCP* протокол је задужен за дефинисање редоследа пакета који стижу (секвенце).

Обзиром да *IP* протокол нема механизам који осигурава поуздан пренос података, процес усмеравања (рутирања) пакета унутар мреже је брз и једноставан.

Уобичајене скраћенице које се користе, а са којима се могу срести програмери при изради интернет апликација, су следећи:

- Слој апликације: *BGP, DHCP, DNS, FTP, HTTP, IMAP, IRC, LDAP, MGCP, NNTP, NTP, POP, RIP, RPC, RTP, SIP, SMTP, SNMP, SOCKS, SSH, Telnet, TLS/SSL, XMPP*
- Транспортни слој: *TCP, UDP, DCCP, SCTP, RSVP, ECN*
- Интернет слој *IP, IPv4, IPv6, ICMP, ICMPv6, IGMP, IPsec*
- Слој везе: *ARP, NDP, OSPF, Tunneling protocol, L2TP, PPP, Media access control, Ethernet, DSL, ISDN, FDDI*

Телнет (*Telnet*) је мрежни протокол унутар групе интернет протокола. Намена је успостављање двосмерног осмобитног комуникационог канала између два умрежена рачунара. Користи се да осигура кориснику једног рачунара сесију за коришћење командне линије на другом рачунару; назив потиче од *Telephone Network*. Телнет је један од најстаријих протокола слоја апликације. Подаци који се размењују нису заштићени и пресретањем се могу протумачени. *SSH* (*Secure Shell*) је наследник Телнета.

Међу посебно важне протоколе апликационог слоја спадају и *DHCP* (порт 67/68) и *DNS* (порт 53) сервери. *DNS* (*Domain Name System*) је систем који претвара имена рачунара (*hostnames*) у *IP* адресе.

DHCP (*Dynamic Host Configuration Protocol*) је протокол за динамичко конфигурисање рачунара. То је скуп правила који омогућава уређајима на рачунарској мрежи да траже и добију *IP* адресу од *DHCP* сервера, да прибаве аутоматски дељену адресу и сазнају

додатне информације као што је адреса његовог рутера за први скок и адреса његовог *DNS* сервера. *DHCP* је у стању да аутоматизује мрежне аспекте, и назива се још *plug-and-play* протокол.

За апликацију електронске поште користе се одвојени сервери за долазну и одлазну пошту. Одлазна пошта користи *SMTP* (*Simple Mail Transport Protocol*) који слуша на порту 25. Најчешће коришћени долазни протокол за електронску пошту је *POP3* (*Post Office Protocol* верзија 3) који користи порт 110. *POP3* служи да преузме мејлове са удаљеног маил сервера и да их смести локално.

1.1.7.1 Рачунарски мрежни порт

Рачунарски мрежи порт је софтверски канал којим комуницирају апликације путем рачунарских мрежа. Софтверски канал је на једној од страна комуникације представљен јединственим бројем који користе протоколи транспортног слоја ОСИ модела у циљу разликовања (раздвајања), идентификације и праћења комуникације апликација. Протокол *TCP* наводи бројеве портова у својим заглављима као изворишни и одредишни порт, како би се при транспорту знало ко је послао и ко треба да прими податке. У клијент-сервер комуникацији, изворишни порт представља број порта који означава апликацију која иницира комуникацију, док одредишни порт означава статички број порта сервиса на серверу. Клијенти динамички бирају број порта за сваку конверзацију. Организација ИАНА (*Internet Assigned Numbers Authority*) је одговорна за доделу стандардних бројева портова.

Бројеви портова најпознатијих придружених апликација или протокола виших нивоа су следећи:

- Статички портови имају опсег од 0 до 1023: 20 и 21 за *FTP*, 23 за *Telnet*, 26 за *SMTP*, 53 за *DNS*, 67 за *DHCP*, 80 за *HTTP*.
- Резервисани портови имају опсег од 1024 до 49151: 1503 за *Windows Live Messenger*, 4224 за *CDP*, 5050 за *Yahoo*! *Messenger*, 5060 за *SIP*.
- Динамички портови имају опсег од 49152 до 65535.

Софтверске портове треба разликовати од хардверских портова. Рачунарски хардверски портови су рачунарски прикључак, конектор или порт интерфејс на рачунару са којим се може повезати неки уређај.

1.1.8 Мрежни хаб, рутер и свич

Постоје три уређаја која се користе за повезивање рачунара, хаб (*hub*), свич (*switch*) и рутер (*router*), које је по некад тешко разликовати само на поглед.

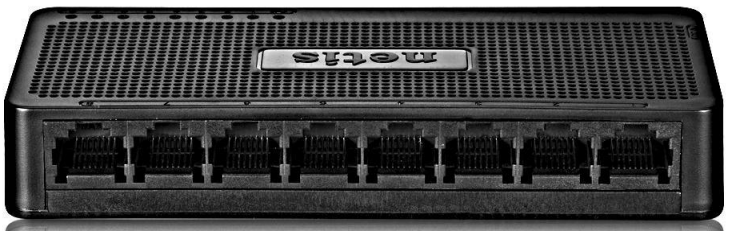
1.1.8.1 Хаб (*Hub*)

Намена мрежног хаба јесте да повеже два или више рачунара без стварног разумевања онога шта се преноси. Уобичајено се користи за приватну мрежу и намењена је за локалне рачунаре. Подразумева се да нема директан приступ Интернету. Када хаб прими пакет података са једног повезаног уређаја, он тај пакет преноси свим другим уређајима са којим је повезан, без обзира коме је намењен садржај. Мрежни пропусни опсег је подељен између свих повезаних рачунара; више повезаних рачунара значи мање брзине преноса

података зато што сви рачунари добијају податке, и они којима су подаци намењени и они којима нису намењени. Хаб може бити повезан на рутер и преко рутера се приступа Интернету. Некада су свичеви и рутери били скупи и компликовани за рад, и тада су хабови били популарни. Данас свичеви и рутери имају приступачну цену, и није их компликовано користити, па се хабови ретко користе у повезивању рачунара.

1.1.8.2 Свич (*Switch*)

Мрежни свич повезује више рачунара као да је мрежни чвор. Када свич прими пакет података од једног рачунара, он на основу физичке *MAC* (*Medium Access Control*) адресе одређује ком рачунару је намењен пакет и шаље податке само том рачунару. Свич не шаље податке свим рачунарима повезаним на свич, па пропусни опсег није затрпан сигналима свих повезаних рачунара, те не мора да се дели пропусни опсег, због чега се и мрежа ефикасније користи. У суштини, пошто се ради о преспајању две везе преко свича, може се користити и термин прекидач. Свич је одличан избор по питању цена-перформансе ако је потребно повезати неколико рачунарских уређаја без потребе да сви рачунари буду повезани на Интернет. У шематским приказима свич се често приказује као диск или квадрат на коме су нацртане стрелице које иду од тачке која је у центру објекта.

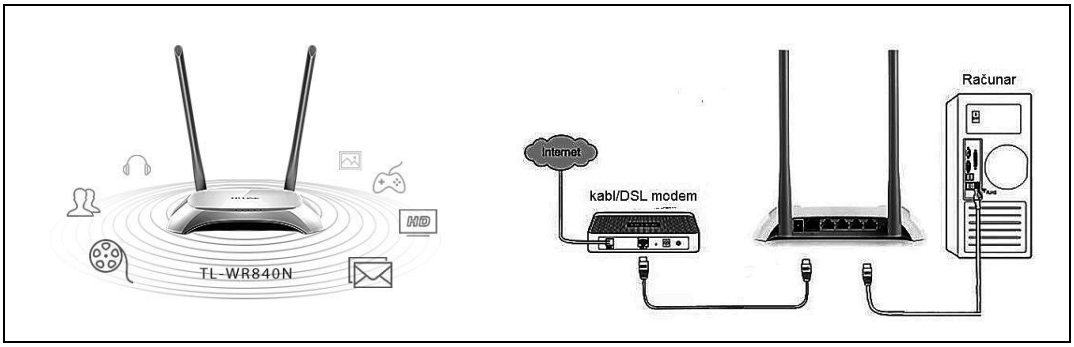


Слика 1.1.3 ST3108S *Fast Ethernet Switch*.

Слика1.1.3 илуструје осмопортни брзи Етернет свич који омогућава да се жично повезана мрежа једноставно прошири на осам портова брзине 10/100Mbps.

1.1.8.3 Рутер (*Router*)

Мрежни рутер се разликује од свича и хаба зато што је његова основна намена да преусмерава пакете података у друге мреже, ка другим рачунарима, користећи различите путеве преноса, односно да податке са једног рачунара, из једне рачунарске мрежа, шаље на Интернет користећи *IP* адресу одредишног рачунара или мреже. На пример, бежични рутер омогућава бежичним уређајима (паметном телефону, таблету, бежичном лаптопу) да се повежу на локалну мрежу а преко ње и на Интернет. Рутер је најбоље решење ако сви рачунари и бежични уређаји треба да буду повезани на Интернет.



Слика 1.1.4 300Mbps *Wireless* рутер (TP-LINK TL-WR840N).

Слика 1.1.4 илуструје бежични рутер велике брзине (до 300Mb/s) који је компатибилан са стандардом *IEEE* 802.*11b/g/n*, који је погодан за *HD* стриминг, онлајн игре и преузимање великих фајлова.

На серверима опремљеним серверским оперативним системима који имају најмање два мрежна прикључка могуће је инсталирати софтверски рутер; сервер одређује рутирање пакета. На пример, на *Windows* 2000 серверу могуће је покренути *Routing and Remote Access* рутирање. У шематским приказима рутер се често приказује као диск на коме су нацртане 4 стрелице од којих две иду ка центру диска, а две од центра диска.

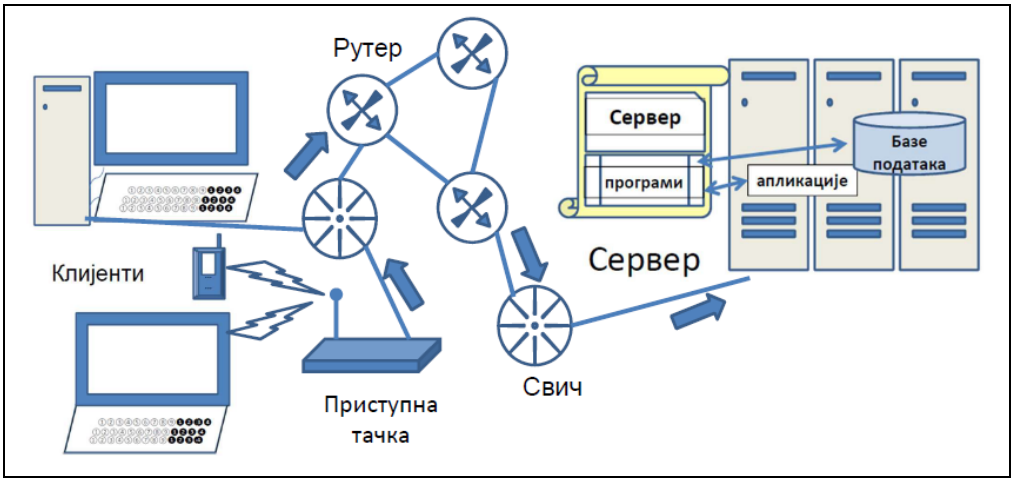
1.1.8.4 Бежична приступна тачка (*WAP*)

Бежична приступна тачка (*Wireless access point - WAP*) је мрежни уређај који спаја *WiFi* бежичне уређаје са жичним мрежама (преко Етернет прикључка) користећи стандарде попут *WiFi* или Блутут. Углавном се повезује са рутером. *WAP* управља преносом на основу избора радио канала и посредује у преносу података између повезаних клијената. Неки рачунари имају уграђени бежични *WiFi* уређаје који могу да користе софтвера за обављање ове функције.

1.1.8.5 Блутут (*Bluetooth*)

Блутут (*Bluetooth*) је технологија помоћу које се врши бежични пренос података између уређаја који поседују исту технологију, на пример између ПДА (*personal digital assistant*) уређаја, мобилних телефона, лаптоп рачунара, класичних рачунара, дигиталних фото апарата и камера.

Слика 1.1.5 илуструје пример архитектуре мреже у којој се показује како је повезан паметни мобилни телефон са сервером. Мобилни телефон остварује бежичну везу са бежичном приступном тачком, па преко ње је повезан са свичем. Свич прослеђује податке до рутера, који има могућност да део поруке проследи до следећег рутера горе десно а део поруке до рутера доле десно. Порука од рутера горе десно долази до рутера доле десно, а обе поруке се преко свича прослеђују серверу. У серверу се примају све приспеле поруке, поруке које су настале сегментацијом на првом рутеру лево се поново спајају у целину, и коначно достављају серверу на анализу и обраду.



Слика 1.1.5 Повезивање клијената са сервером преко приступне тачке, свичева и рутера.

Сервер обрађује поруку добијену од клијента, формира одговор на захтев клијента, и шаље преко свича до рутера доле десно. Рутер истражује оптималан пут до клијента и шаље на следећи рутер. Када серверова порука стигне до рутера десно, овај рутер је шаље до свича. Свич анализира адресу одредишта, и шаље ка приступној тачки, одакле одговор коначно стиже до мобилног телефона.

Преко исте приступне тачке може да се остварује комуникација и са мобилним телефоном и са лаптопом. Такође, преко истог свича, сви могу да остварују конекцију са рутером, па преко других рутера до десног свича и сервера.

Преостаје да се сагледа како поруке знају коме су упућене и како оне те информације преносе свичевима и рутерима.

1.1.9 *IP* адреса

IP адреса (*Internet Protocol address*) је јединствена адреса (условно речено) уређаја на Интернету, односно, јединствени број, сличан телефонском броју, који користе рачунари у међусобном саобраћају путем Интернета уз коришћење интернет протокола. *IP* (*Internet Protocol*) представља скуп правила који сваки рачунар, таблет, мобилни телефон поштује на Интернету. *IP* адреса је попут поштанске адресе, која дефинише државу, град, улицу и број на јединствен начин на свету. *IP* адреса се састоји од четири броја раздвојена тачкама, и има облик #.#.#.#, где сваки знак # има вредност која се налази на интервалу 0-255. Овакав запис *IP* адресе се назива децимални. Сваки број у *IP* адреси користи 8 бита (који се називају октети), што значи да адреса укупно има 32 бита. Ово дозвољава рачунарима да прослеђују информације у име пошиљаоца (како би рачунари знали где да их даље пошаљу) и касније примање тих информација (како би рачунари знали да је то одредишна дестинација). Пример *IP* адресе је 51.254.100.34. У овом тексту, под појмом рачунар се посматра у ширем смислу сваки уређај који шаље и прима податке од других уређаја повезаних у рачунарску мрежу.

Могуће је направити укупно 2^{32} = 4 294 967 296 комбинација са 32 бита као могуће адресе свих који су повезани на Интернет. Иако може да се направи око 4 милијарде различитих адреса, постоји огроман број уређаја на Интернету, тако да број различитих адреса постаје недовољан чак и за овако велику вредност. Неки опсези адреса су додељени организацијама или провајдерима, па је број расположив за адресирање и мањи од тог броја. Светска организација која брине о глобалној координацији адреса је *IANA* (*Internet Assigned Numbers Authority*). *IP* адресе које почињу са 10.#.#.#, 172.16.#.# - 172.31.#.#, или 192.168.#.# представљају приватне *IP* адресе које се могу користити у оквиру неке локалне мреже, али не и на Интернету. Конвертовање у ове бројеве из за људе читљивије форме адреса домена попут www.wikipedia.org, се врши путем *DNS*-а. Процес конверзије је познат под именом растављање имена домена.

Проналажење адресе рачунара са кога се приступа Интернету као клијент, и адресе коју желимо да пронађемо на Интернету, адресу сервера, може да се уради коришћењем команде *nslookup* (са рачунарима на којима је Мајкрософт оперативни систем). На пример, стартовањем апликације *Command Prompt*, отвара се текстуални приступ радном директоријуму **`C:\Users\MIROSLAV>`** где се после симбола **`>`** уноси команда **`nslookup yahoo.com`**, која проверава да ли постоји веза са сајтом *yahoo.com*:

```
C:\Users\MIROSLAV>nslookup yahoo.com
Server:    cns3.vektor.net
Address:   109.122.98.6
Non-authoritative answer:
Name:      yahoo.com
Addresses: 2001:4998:c:1023::5
           2001:4998:c:1023::4
           2001:4998:44:41d::4
           2001:4998:58:1836::11
           2001:4998:44:41d::3
           2001:4998:58:1836::10
           98.137.246.7
           98.138.219.232
           98.137.246.8
           72.30.35.9
           98.138.219.231
           72.30.35.10
C:\Users\MIROSLAV>
```

Као одговор, добијају се подаци од *DNS* сервера са које адресе се приступа Интернету, (**`Server: cns3.vektor.net, Address: 109.122.98.6`**), што је у овом случају адреса кабловског оператера коме се приступио са клијентовог лаптопа. Од *DNS* сервера се добијају назив и адресе сервера (**`Name: yahoo.com, Addresses:`** ...). Када се приступу са рачунара који је у ВИШЕРу, добија се другачије име радног директоријума, и назив сервера, као и другачија адреса сервера:

```
C:\Users\korisnik>nslookup yahoo.com
Server:  unity.vets.edu.yu
Address:  172.16.1.1
```

Сви подаци о серверу су исти у оба приступа.

Ако преко кабловског оператера потражимо податке о другом серверу (**nslookup viser.edu.rs**) добија се иста адреса са које тражимо информације, али другачији подаци о серверу:

```
C:\Users\MIROSLAV>nslookup viser.edu.rs
Server:  cns3.vektor.net
Address:  109.122.98.6
Non-authoritative answer:
Name:    viser.edu.rs
Address:  147.91.194.12
C:\Users\MIROSLAV>
```

Симболичке адресе или имена сервера су природније када их користе људи (**Name: viser.edu.rs**), а за рачунаре је једноставније да користе адресу бројевима (**Address: 147.91.194.12**).

У веб прегледачу могу да се унесу имена сервера (viser.edu.rs) или адресе (147.91.194.12), и у оба случаја треба очекивати да се добије идентичан садржај са сервера.

Они који више воле да добију информације преко веб прегледача, могу да користе овај сервер тако што у поље за приступ серверу на клијентском прегледачу унесу
//ping.eu/traceroute/,
а затим у поље за *IP* адресе коју траже назив сервера,
viser.edu.rs.

У *Command Prompt*–у може се стартовати команда
tracert viser.edu.rs,

и добија се сличан садржај.

За хардверско повезивање уређаја користи се физичка адреса коју дефинише произвођач на основу *MAC* (*Medium Access Control*) адресе. *MAC* адреса се састоји од 48 бита, при чему првих 24 бита служе да означе произвођача, док преосталих 24 бита одређује сам уређај одређеног произвођача. *MAC* адресе се уобичајено представљају у хексадецималном запису (цифре од 0-9 и латинична слова *A, B, C, D, E, F*).

1.1.10 Питања и задаци за вежбу

1. Отворити Команд промпт (добија се прозор са црном позадином) и пронаћи адресу неког сервера, на пример ВИШЕР (**viser.edu.rs**).
2. Користећи команду **tracert**, пронаћи којим путем се успоставља конекција клијентског рачунара и сервера, и илустровати временима која се добијају као одговор.
3. Користећи адресу сајта **//ping.eu/traceroute/**, у веб прегледачу, пронаћи којим путем се остварује веза са сервером на адреси **viser.edu.rs**. Упоредити времена са оним која су добијена од команд промпта.

1.2 Оперативни систем и *HTML*

Резиме

Циљ овог поглавља је да студент разуме софтверски део архитектуре која се користи за веб апликације. Дати су најосновнији појмови за оперативне системе, веб-прегледаче, едиторе текста, фајловове са екстензијом *html*, форматирање засновано на *CSS*-у, Јава програми, *CMD, Command Prompt*.

Увод у оперативни систем и *HTML*

Претходно поглавље садржи основне појмове о инфраструктури на којој се заснива интернет програмирање, како би студенти имали све појмове који се касније појављују у оквиру овог курса, без потребе да објашњења траже у другим изворима из ове области. У овом поглављу студенти се упознају са основним елементима оперативних система и градивом које ће бити коришћено из *HTML*. Из области оперативних система обрађују се оне операције и команде које су неопходне за инсталирање сервера и преименовање и премештање фајлова на меморијске локације како би инсталирани програми и програми који се развијају били тачно тамо где је потребно да би све функционисало без грешака. На пример, за развој програма који се налазе на страни сервера, програмирање се може обавити на клијентском рачунару где је могуће одмах проверити исправност функционисања програма и пре него што се програми поставе на сервер. Такође, када се користе изворни Јава програми и њихове класе добијене компајлирањем, програми се неће налазити у директоријумима како то очекују они који већ имају искуства са Јава програмирањем. У сродној литератури се може наћи како да се нешто уради, али се ретко могу пронаћи подаци о томе где се налазе фајлови, што програмерима и корисницима апликација може бити велики проблем. У интернет апликацијама се фајлови налазе у тачно одређеним фолдерима, а често се користе и алати за развој који олакшавају практичну имплементацију знања из ове области. У неким случајевима је потребно урадити и системска подешавања, па ће и о томе бити речи у овом поглављу.

Програмирање интернет апликација у овом курсу се састоји у размени фајлова између клијента и сервера, при чему програми на серверу треба да генеришу фајлове који су у *HTML* формату. То конкретно значи, у случају Јава сервлета, да Јава програм који програмер треба да напише и постави на серверу, у ствари генерише текстуалну датотеку у *HTML* формату, тако да прегледач на страни клијента може да интерпретира и прикаже садржаје стандардним веб прегледачима. Обзиром да се информације убацују у *HTML* фајл, за праћење излагања је неопходно добро познавање *HTML*. Стога ће у овом поглављу бити поновљене елементарне карактеристике *HTML*, а више детаља о *HTML* студенти могу наћи у другим курсевима. Такође, више детаља о Јаваскрипу и Јави студенти могу да нађу у оквиру других предмета на овим студијама. Да би избегли непотребно тражење из друге литературе, овде ће бити поновљени само они детаљи који се користе за потребе овог предмета (Интернет програмирање).

Већина упутстава за инсталирање софтвера који се користи за интернет програмирање ради на различитим хардверским уређајима, са различитим оперативним системима, па ће у овом поглављу бити наведени алтернативни називи који се разликују од рачунарског

система који је намењен апликацијама. Осим текстуалних упутстава постоје и снимљене видео секвенце за убрзано учење и пожељно је да програмери знају и алтернативне називе појмова.

1.2.1 Оперативни систем

Као што се користи интернет адреса, као назив састављен од слова и бројева, које људи могу разумети да би добили информације са неког сервера на Интернету (на пример **`viser.edu.rs`**), а посебни сервиси (*DNS*) претварају ту адресу у бројеве које користе рачунарски уређаји за комуникацију (на пример **`147.91.194.12`**), тако постоји више начина да корисник комуницира са рачунаром а што се разликује од начина комуникације два рачунара. Првобитни рачунари су омогућавали готово искључиво текстуални начин задавања наредби рачунару, док је код савремених рачунара олакшана комуникација кликом миша на слику команде или притиском функцијских тастера и истовременим притиском више тастера са тастатуре.

Интерфејс командне линије (*command-line interface, CLI*) користи се да човек унесе команде за извођење специфичних задатака и интеракцију са оперативним системом или неким специфичним софтвером рачунара. У првој теми је наведена једна таква команда `nslookup`, која проверава да ли постоји веза са неким сајтом.

Овакав начин уношења команди рачунару се називају на енглеском језику *command-line interface, command language interpreter, shell, command-line user interface, console user interface, character user interface*. Суштина је да се уноси текстуална порука састављена од слова, бројева и симбола у једној или више линија (такозване командне линије, *command lines*), где се свака линија завршава знаком преласка у следећу линију. Под појмом интерпретер командне линије подразумева се да рачунар прима сет инструкција у једној линији, анализира их и извршава захтевани сет команди. Рачунар интерпретира сваку линију одмах након детектовања краја линије (крај линије се често назива *newline, line ending, end of line, EOL, line feed, LF, line break*, који се уноси типком *Enter key*). *ASCII* кодна ознака је *CR* и *LF control codes*, а користе се и ознаке *next line, NEL control code, control codes for line separator* као и *paragraph separator markers*. Који ће се контролни знак или карактер користити зависи од тога ко је направио рачунар и од оперативног система рачунара. За потребе писања програма који се извршавају аутоматски или као текст у више линија, користи се и такозвана ескејп секвенца (*escape sequence*), која има на пример неколико карактеристичних ознака: *line feed* је \n, *carriage return* је \r, *tabulator* је \t.

Искусни програмери више воле да раде користећи интерфејс командне линије, где у посебном прозору уносе команде које рачунар треба да изврши, и у истом прозору рачунар исписује, линију по линију, резултате извршавања команди или извештај о завршетку извршавања наредбе. Међутим, већина корисника који нема програмерско искуство више воли да користи графички кориснички интерфејс за комуникацију са рачунаром и апликацијама (*Graphical User Interface, GUI*). Цео сет инструкција може да се сакрије графичким симболом, који интуитивно подсећа на то шта треба да се уради, и довољан је клик миша на сличицу да би се извршиле сложене команде. На пример, један начин јесте да се у командној линији исписује команда за штампање текста заједно са описом ком принтеру или хардверском порту се шаље текст за штампање и евентуално додатна подешавања принтера. Алтернативно, кликом на иконицу која подсећа на

принтер, и потврдом да текст треба да се штампа на подразумеваном принтеру са унапред подешеним параметрима, клијенти лако могу да обаве задатак штампања.

За оне који се баве програмирањем интернет апликација, неопходно је познавање и уношења команди текстуално у посебном прозору, али исто тако своје апликације треба да направе да буду прилагођене клијентима који имају недовољно информатичко знање и да интуитивно кликтањем по сличицама и дугмићима обављају сложене операције.

У рачунарству, оперативни систем је скуп команди које контролишу и управљају рачунарским компонентама и обављају све операције који се односе на делове рачунара. Иако корисник рачунара често не види како све ово функционише, програми који се инсталирају на рачунару могу преко оперативног система да комуницирају са деловима рачунара. Мрежни оперативни систем је другачија врста оперативног система.

Апликације и програми који су инсталирани на рачунару добијају инструкције и податке од корисника рачунара преко одговарајућег интерфејса, а затим те апликације позивају команде оперативног система које даље управљају рачунаром. Оперативни систем је на пример посредник између апликација и меморија рачунара, алоцира где ће бити смештени подаци апликација у меморији рачунара, који подаци меморије рачунара треба да буду пренети на екстерне уређаје преко хардверских портова и како се подаци обрађују и шта се са подацима дешава након обраде.

Најпознатији оперативни системи су *Microsoft Windows, macOS* и *Linux*. Код мобилних телефона, најпознатији су *Android, Apple iOS* и *Linux*.

Након инсталирања оперативног система на рачунару, могу се инсталирати апликације које ће користити клијенти, као што су едитори текста или веб прегледачи.

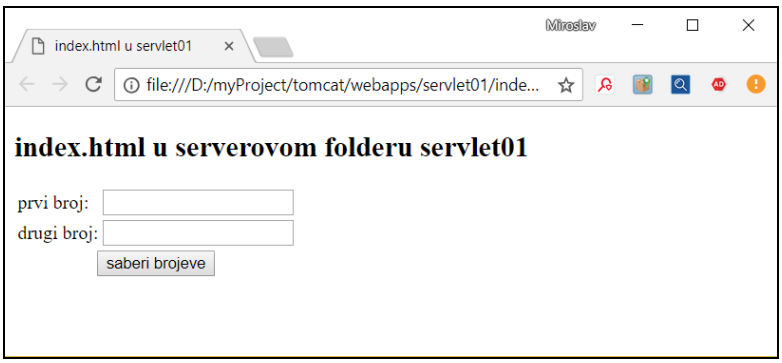
1.2.2 Веб-прегледач

Апликација која се назива Веб-прегледач (*web browser*) омогућава гледање докумената који су у *HTML* формату, без обзира да ли су направљени на самом рачунару, или су преузети са неког другог рачунара или са сервера уношењем *IP* адресе сервера (*URL, Uniform Resource Locator*). *HTML* документ може да садржи и мултимедијалне садржаје (слика, видео снимак, и музички запис). Најпознатији прегледачи су *Google Chrome, Mozilla Firefox, Internet Explorer, Safari, Microsoft Edge, Opera*. Веб прегледач је кључна апликација која се користи у Интернет програмирању.

1.2.3 Едитор текста

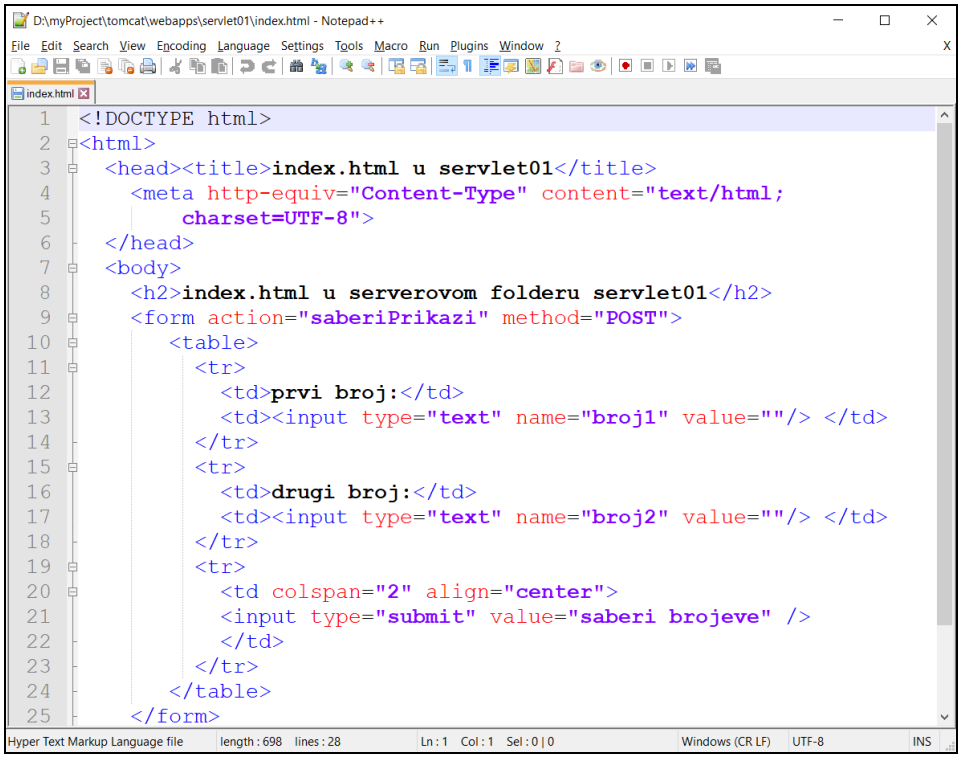
За писање програма и докумената као што су *HTML*, програмери могу да користе програме који су посебно намењени за такве сврхе. Програм Ноутпед (*Notepad*) је најједноставнији уређивач текста који се испоручује са Мајкрософт системима. Иако већина користи програме за обраду текста вишег нивоа као што је у оквиру програмског пакета Мајкрософт офис програм Мајкрософт Ворд, који је типа визивиг (*wysiwyg - What You See Is What You Get* - шта видиш то и добијаш), ови програми нису погодни за писање рачунарских програма. Програми за уређење теста вишег нивоа приказују како ће текст изгледати када се одштампа, и има сакривене делове кода који се користе за приказ одштампаног текста. За писање рачунарских програма ове додатне сакривене инструкције могу да доведу до нежељених ефеката, односно, програм написан у Ворду неће радити. Из

класе програма као што је Ноутпед, препоручујемо да се користи бесплатни софтвер (софтвер који се може користити без плаћања лиценце за коришћење) *Notepad++*.



Слика 1.2.1 Приказ *HTML* фајла у прегледачу *Chrome*.

Слика 1.2.1 илуструје пример фајла *index.html* у прегледачу који би се налазио на серверу, а коме би се приступало преко адресе сервера; у следећем примеру се фајл налази на D диску рачунара **`file:///D:/myProject/tomcat/webapps/servlet01/index.html`**.



Слика 1.2.2 Део кода у едитору текста *Notepad++* за *HTML* фајл приказан на Слици 1.2.1.

На Слици 1.2.2 је приказан пример фајла *index.html* написаног у програму *Notepad++*. Основна предност *Notepad++* програма јесте да препознаје карактеристичне речи које су команде а које се користе у рачунарском коду и означава их различитим бојама, и са леве стране се могу уочити који делови кода су целине. Захваљујући овим особинама, и пре свега означеним бројевима линија, лако се идентификује где је који део софтверског кода. Током тестирања кода, уколико постоје грешке у коду, програми за тестирање ће јавити у којој линији и којој колони је настала грешка, која команда је погрешно написана или који део кода није затворен одговарајућом заградом или тагом.

Иако има оних који сматрају да *HTML* није програм, *HTML* има команде које прегледач интерпретира и на основу њих приказује садржаје у прегледачу. *HTML* фајлови имају прецизно дефинисану структуру као што се може видети на Слици 1.2.2.

За потребе овог курса, едитор текста *Notepad++* биће основни едитор у коме ће бити написани *HTML* фајлови, али и сви остали изворни програми у програмским језицима, или као опис параметара који се користе за подешавање рада у комуникацији између клијента и сервера.

1.2.4 Фајлови *HTML*

На слици 1.2.2 је приказан пример *HTML* фајла. *HTML* се учи као посебан курс, а у овом поглављу ће бити наведене само основне карактеристике за које се подразумева да их студент већ добро познаје. Сврха је да се студент подсети на основне инструкције и правила *HTML* како би пратио наредна поглавља без потребе да користи другу литературу.

HTML (*HyperText Markup Language*) је описни језик намењен опису веб страница коју прегледачи треба да прикажу. *HTML* је стандардизован језик у који су уграђени елементи који детаљније описују сам документ. Код који је приказан на слици 1.2.2 а који прегледач приказује на монитору као на слици 1.2.1 састоји се од следећих линија кода:

```
<!DOCTYPE html>
<html>
  <head><title>index.html u servlet01</title>
       <meta http-equiv="Content-Type" content="text/html;
            charset=UTF-8">
  </head>
  <body>
    <h2>index.html u serverovom folderu servlet01</h2>
       <form action="saberiPrikazi" method="POST">
          <table>
            <tr>
              <td>prvi broj:</td>
              <td><input type="text" name="broj1" value=""/> </td>
            </tr>
            <tr>
              <td>drugi broj:</td>
              <td><input type="text" name="broj2" value=""/> </td>
            </tr>
            <tr>
              <td colspan="2" align="center">
              <input type="submit" value="saberi brojeve" />
              </td>
            </tr>
          </table>
       </form>
</body>
</html>
```

Сви *HTML* документи би требало да почињу са дефиницијом типа документа (*DTD, Document Type Definition*) који прегледачу преносе информацију по ком стандарду је документ писан како би се избегле грешке у интерпретацији. На пример:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

Претходни део кода преноси прегледачу да је документ писан по строгом HTML 4.01 стандарду.

У нашем примеру је само информација да се ради о **`html`** документу:

```
<!DOCTYPE html>
```

HTML документ има две основне целине:

1. Информације које описују документ налазе се између ознака за заглавље документа **`<head>`** и **`</head>`**; у оквиру заглавља поставља се посебно одвојена целина која се користи за наслов документа, и она је између ознака **`<title>`** и **`</title>`**
2. Садржај *HTML* смешта се између ознака **`<body>`** и **`</body>`**, и овај део се интерпретира од стране прегледача да би био приказан на екрану посетиоца сајта.

Први део описује документ и начин на који треба да буде представљен. Други део представља садржај документа. Ова три елемента заједно са ознакама **<html>**, **</html>** чине минималну структуру *HTML* документа.

У нашем примеру је у заглављу

```
<head><title>index.html u servlet01</title>
     <meta http-equiv="Content-Type" content="text/html;
          charset=UTF-8">
</head>
```

Наслов документа се види у врху језичка прегледача **index.html u servlet01** на слици 1.2.1. Мета подаци (невидљиви међу-подаци) се користе да се додатно опише сам документ и не приказују се на монитору прегледача.

Треба уочити да су целине кода увучене (индентоване) како би се лакше уочиле и пратиле целине. Није потребно увлачити текст због приказа садржаја у прегледачу, и белине настале куцањем бланко знака (дугачки тастер у доњем делу тастатуре) немају никакав утицај на приказ у прегледачи. Једина сврха увлачења линија текста и убацивања бланко знакова јесте да се текстуални опис *html* документа учини прегледнијим.

Основ *HTML* језика су тагови (*tags*) и атрибути. Тагови се постављају између знакова за мање и веће "**<XXX>**". Ако уместо **XXX** стоји неки текст који прегледач разуме шта треба да уради, онда ће тако и бити интерпретиран текст који се налази после тага све до краја документа. Када се жели да само део документа буде интерпретиран на одређени начин, онда се на крај тог дела текста поставља завршни таг који има косу црту (**/**), на пример "**</XXX>**", као ознака да се остатак текста више не интерпретира на начин како је почетни таг "**<XXX>**" одредио.

У нашем узорном коду уместо **XXX** користи се **h2**

```
<h2>index.html u serverovom folderu servlet01</h2>
```

То значи да се текст између ових тагова исписује врстом слова која су дефинисана за наслове нивоа 2 (**h** је *header tag*, а број означава величину слова која је дефинисана за овај таг, односно *header tag* **h** је ознака за таг наслова).

Унутар тагова се могу налазити атрибути који додатно дефинишу шта треба да уради прегледач. На пример

```
<td colspan="2" align="center">
```

одређује да ширина поља у које се уноси текст заузима две колоне и да текст треба да буде постављен у средину поља; у ствари, ово је дугме које се касније може кликнути мишем да би био послат резултат збира два броја.

Атрибути се налазе унутар сложених тагова и садрже додатне информације о начину приказивања и понашању означеног дела документа. Вредности атрибута (**"2"** и **"center"** се смештају између знакова навода. Веб прегледач игнорише непознате атрибуте и елементе и не прави разлику између великих и малих слова (*case insensitive*). Прегледач аутоматски прелама текст према ширини прозора у коме приказује садржај и прави нове редове када је потребно.

За потребе интернет програмирања, користићемо друге програме да направимо *HTML* документе које ће клијентски прегледачи преузимати и обављати програмиране активности. Другим речима, *HTML* документ је основни тип документа и програмер мора да има солидно знање о *HTML* да би могао да напише исправан програм.

1.2.4.1 *HTML* тагови

Стандаризоване верзије су биле *HTML* 2.0 (*RFC* 1996), *HTML* 3.2 (1997), *HTML* 4.0 (1997), *HTML* 4.01 (1999), *ISO HTML* 4.01 *Strict* (2000), *HTML* 5 (2008). Тренутно актуелне верзије које се користе у пракси су *HTML* 4.01 и *HTML* 5.2 (2018).

У наставку су дати тагови који се користе у актуелним верзијама *HTML*, или коментари шта да се користи, при чему су са 5 означени тагови који постоје у *HTML* 5 а не постоје у *HTML* 4, а са ~~5~~ означени тагови који постоје у *HTML* 4 а не постоје у *HTML* 5:

<!--...-->	коментар
<!DOCTYPE>	тип документа
<a>	хиперлинк
<abbr>	скраћеница или акроним
<acronym> ~~5~~	акроним
<address>	контакт информација о аутору или власнику документа
<applet> ~~5~~	уграђени аплет, користити <embed> или <object> у 5
<area>	површина у *image-map*
<article> 5	независни, самостални садржај
<aside> 5	садржај поред веб странице
<audio> 5	звук
<b>	подебљани текст
<base>	основни *URL* за документ
<basefont> ~~5~~	користити *CSS* за подразумевану боју, величину и тип слова за документ
<bdi> 5	издвојени део који може да се форматира другачије од осталог текста
<bdo>	предефинисање форматирање текста
<big> ~~5~~	користити *CSS* за велика слова за документ
<blockquote>	секција која се цитира из другог извора
<body>	тело документа
 	нови ред
<button>	дугме које може да се кликне
<canvas> 5	цртање графика, *on the fly*, *via scripting* (*usually JavaScript*)
<caption>	наслов табеле
~~5~~	користити *CSS*, центрирани текст
<cite>	наслов неког дела
<code>	део рачунарског кода
<col>	специфицирање особина колона за сваку колону у <colgroup> елементу
<colgroup>	група колона у табели са форматирањем
<data> 5	линковање одређених података са машински читљивим преводом
<datalist> 5	специфицира листу пре-дефинисаних опција за улазну контролу
<dd>	опис/вредност термина у дескриптивној листи
<del>	текст који је обрисан у документу (прецртано)
<details> 5	додатни детаљи које корисник може да види или сакрије

<dfn>	инстанца термина
<dialog> 5	прозор или бокс
<dir> ~~5~~	листа директоријума; користити
	секција у документу
<dl>	дескриптивна листа
<dt>	термин/име описне листе
	истакнути текст
<embed> 5	контејнер за екстерну апликацију која није *HTML*
<fieldset>	група елемената у форми
<figcaption> 5	наслов <figure> елемента
<figure> 5	специфицира засебан садржај
<font> ~~5~~	тип, боја и величина слова; користити *CSS*
<footer> 5	подножје (*footer*) документа или секције
<form>	*HTML* форма за унос података
<frame> ~~5~~	прозор или фрејм
<frameset> ~~5~~	сет фрејмова
<h1> ... <h6>	*HTML* наслов
<head>	информације о документу
<header>	заглавље документа или секције
<hr>	тематска промена у садржају
<html>	почетак *HTML* документа
<i>	укошен тип слова
<iframe>	фрејм у истој линији
<img>	слика
<input>	контрола уноса
<ins>	текст унет у документ (подвучено)
<kbd>	унос са тастатуре
<label>	ознака <input> елемента
<legend>	ознака <fieldset> елемента
	ставка у листи
<link>	повезивање са екстерним ресурсом или стилом
<main> 5	главни садржај документа
<map>	мапирање дела слике у површ на коју се кликне (*client-side image-map*)
<mark> 5	маркиран / истакнут текст
<meta>	метаподаци о *HTML* документу
<meter> 5	скаларна мера у задатом опсегу (приказ измерене вредности)
<nav> 5	навигациони линк
<noframes> ~~5~~	алтернативни садржај за кориснике који не подржавају фрејмове
<noscript>	алтернативни садржај који не подржавају скрипт на клијент страни
<object>	уграђени (*embedded*) објекат
	нумерисана листа
<optgroup>	група повезаних опција у *drop-down* листи
<option>	опција у *drop-down* листи
<output> 5	резултат израчунавања
<p>	параграф
<param>	параметар објекта

<picture> 5	контејнер за вишеструки извор слика
<pre>	нови формат текста
<progress> 5	приказује који део задатка је извршен
<q>	кратки наводници
<rp> 5	шта се приказује у прегледачу а није подржано *ruby* означавањем
<rt> 5	објашњење / изговор карактер (за источно-азијску типографију)
<ruby> 5	*ruby* означавање (за источно-азијску типографију)
<s>	текст који није коректан (прецртан текст)
<samp>	дефинише узорак (*sample output*) из рачунарског програма *program*
<script>	скрипта на клијентској страни
<section> 5	секција у документу
<select>	*drop-down* листа
<small>	текст смањене величине
<source> 5	вишеструки извор за *media* елементе (<video> и <audio>)
	секција у документу
<strike> 5	користити <del> или <s>. Прецртани текст
	важан текст (повећан и подебљан текст)
<style>	информације о стилу у документу
<sub>	текст као индекс
<summary> 5	видљив *heading* за детаље <details> елемент
<sup>	текст као експонент
<svg>	контејнер за *SVG* графику
<table>	табела
<tbody>	груписан садржај табеле
<td>	ћелија у табели
<template> 5	дефинише шаблон који може да се приказује коришћењем *JavaScript*
<textarea>	површина са текстом у више редова
<tfoot>	подножје за груписање садржаја у табели
<th>	ћелија заглавља у табели
<thead>	заглавље за груписање садржаја у табели
<time> 5	датум / време
<title>	наслов документа
<tr>	врста у табели
<track> 5	медија елемент са више титлова (<video> и <audio>)
<tt> 5	користити *CSS*; *teletype* текст
<u>	подвучен текст
	ненумерисана листа
<var>	променљива
<video> 5	видео или филм
<wbr> 5	могућ прелом линије

Више детаља може се наћи у уџбенику који је написао професор др Зоран Ћировић, Увод у Интернет технологије, ВИШЕР, 2015 [2015_Cirovic_UuIT].

1.2.5 Форматирање засновано на *CSS*

У оквиру *HTML* документа може да се користи *CSS* (*Cascading Style Sheets*) да би се смањило непотребно исписивање идентичног кода на различитим страницама а у циљу израде сајта који има препознатљив изглед. Наиме, ако би свака страница имала идентичан код, тада би измена на једној страни требало да се уради и на свим другим страницама ако се жели изглед свих страница на сајту као да су рађене по истом узорном документу. Фајлови *CSS* се чувају на једном месту, па једна промена у неком фајлу би се појавила у свим документима на идентичан начин. Овакав приступ значајно олакшава одржавање веб страница сајта.

Радна група за *CSS* је почела да ради на проблемима који нису били обухваћени верзијом *CSS*1, тако да је 1997. развијена верзија *CSS*2. Од 1998. се развија *CSS*3. При избору својстава у веб пројекту увек треба да се провери за циљне прегледаче и оперативне система која верзија ради исправно.

У наставку су дате својства која се користе у актуелним верзијама (верзија *CSS*3):

align-content	Одређује поравнање између линија унутар флексибилног контејнера када ставке не користе сав расположиви простор
align-items	Одређује поравнање за ставке унутар флексибилног контејнера
align-self	Одређује поравнање за одабране ставке унутар флексибилног контејнера
all	Ресетује сва својства (осим *unicode-bidi* и праваца)
animation	Скраћено својство за сва својства анимације
animation-delay	Одређује кашњење почетка анимације
animation-direction	Одређује да ли ће се анимација одвијати у напред, у назад или алтернативним циклусима
animation-duration	Одређује колико дуго ће трајати један циклус анимације
animation-fill-mode	Одређује стил елемента када се анимација не извршава (пре него што стартује, након што се заврши или и једно и друго)
animation-iteration-count	Одређује колико пута анимација треба да се изврши
animation-name	Одређује назив за *@keyframes* анимацију
animation-play-state	Одређује да ли се анимација извршава или је заустављена
animation-timing-function	Одређује криву брзине за анимацију
backface-visibility	Дефинише да ли ће задња страна елемента бити видљива када је окренута према кориснику
background	Скраћено својство за сва својства *background*
background-attachment	Поставља да се позадинска слика помера са остатком странице или је фиксна
background-blend-mode	Одређује режим мешања за сваки слој позадине (боја/слика)
background-clip	Дефинише удаљеност позадине (боје или слике) у односу на елемент
background-color	Одређује позадинску боју елемента
background-image	Одређује једну или више позадинских слика за елемент
background-origin	Одређује почетну позицију позадинске слике
background-position	Одређује позицију позадинске слике
background-repeat	Поставља како ће се позадинска слика понављати
background-size	Одређује величину позадинских слика

border	Скраћено својство за *border-width*, *border-style* и *border-color*
border-bottom	Скраћено својство за *border-bottom-width*, *border-bottom-style* и *border-bottom-color*
border-bottom-color	Поставља боју доње ивице
border-bottom-left-radius	Дефинише заобљење ивице у доњем левом углу
border-bottom-right-radius	Дефинише заобљење ивице у доњем десном углу
border-bottom-style	Поставља стил доње ивице
border-bottom-width	Поставља ширину доње ивице
border-collapse	Поставља да је гранична линија једнострука или двострука
border-color	Поставља боју 4 границе
border-image	Скраћено својство за сва својства граница-слика
border-image-outset	Одређује износ за који се подручје границе слике протеже изван граничног оквира
border-image-repeat	Одређује да је граница слике поновљена, заокружена или растегнута
border-image-slice	Одређује начин исецања границе слике
border-image-source	Одређује путању до слике да се користи као граница
border-image-width	Одређује ширину границе слике
border-left	Скраћено својство за сва својства леве границе
border-left-color	Поставља боју леве границе
border-left-style	Поставља стил леве границе
border-left-width	Поставља ширину леве границе
border-radius	Скраћено својство за својства 4 границе -заокружење
border-right	Скраћено својство за сва својства десне границе
border-right-color	Поставља боју десне границе
border-right-style	Поставља стил десне границе
border-right-width	Поставља ширину десне границе
border-spacing	Поставља растојање између граница суседних ћелија
border-style	Поставља стил 4 границе
border-top	Скраћено својство за *border-top-width*, *border-top-style* и *border-top-color*
border-top-color	Поставља боју горње границе
border-top-left-radius	Дефинише заобљење границе горњег левог угла
border-top-right-radius	Дефинише заобљење границе горњег десног угла
border-top-style	Поставља стил горње границе
border-top-width	Поставља ширину горње границе
border-width	Поставља ширину 4 границе
bottom	Поставља позицију елемента од дна надређеног елемента
box-decoration-break	Поставља понашање позадине и границе елемента на прелому стране, или, за *in-line* елементе (елементе у истој линији) на прелому линије
box-shadow	Додаје једну или више сенчења елемента
box-sizing	Дефинише израчунавање ширине и висине елемента: да ли се укључују допуна и границе, или не
break-after	Одређује понашање прелома стране, колоне или региона након генерисаног оквира (боксу)
break-before	Одређује понашање прелома стране, колоне или региона пре генерисаног оквира (боксу)

break-inside	Одређује понашање прелома стране, колоне или региона у генерисаном оквиру (боксу)
caption-side	Одређује позицију наслова табеле
caret-color	Одређује боју курсора (знака за унос) у пољу за унос, текстуалном региону или било који елемент који може да се едитује
@charset	Одређује кодну комбинацију за слова која се користи, нпр. "*UTF*-8"
clear	Одређује на којој страни покретног елемента, елементи не могу да буду покретни
clip	Одсеца апсолутно позиционирани елемент
color	Поставља боју текста
column-count	Одређује број колона елемента на које треба да се подели
column-fill	Одређује начин попуњавања колона, да ли су балансиране или не
column-gap	Одређује процеп између колона
column-rule	Скраћено својство за сва својства *column-rule*
column-rule-color	Одређује правило за боју између колона
column-rule-style	Одређује правило за стил између колона
column-rule-width	Одређује правило за ширину између колона
column-span	Одређује преко колико колона може да се простире елемент
column-width	Одређује ширину колона
columns	Скраћено својство за *column-width* и *column-count*
content	Користи се са :*before* и :*after* псеудо елементима, за унос генерисаног садржаја
counter-increment	Повећава или смањује вредност једног или више *CSS* бројача
counter-reset	Креира или ресетује један или више *CSS* бројача
cursor	Одређује приказивање курзора миша када је постављен изнад елемента
direction	Одређује правац текста /правац писања
display	Одређује начин приказа *HTML* елемента
empty-cells	Одређује да ли се приказују границе, или се не приказују, и позадину празне ћелије у табели
filter	Дефинише ефекте (на пример, замућење или промена боје) на елементу пре него што се елемент прикаже
flex	Скраћено својство за својства *flex-grow*, *flex-shrink*, и *flex-basis*
flex-basis	Одређује *initial length of a flexible item*
flex-direction	Одређује *direction of the flexible items*
flex-flow	Скраћена особина за *flex-direction* и *flex-wrap properties*
flex-grow	Одређује колико може да се повећа релативно у односу на остатак
flex-shrink	Одређује колико може да се смањи релативно у односу на остатак
flex-wrap	Одређује да ли флексибилну целину треба преломити или не
float	Одређује да ли ће бокс бити слободан или фиксиран
font	Скраћено својство за *font-style*, *font-variant*, *font-weight*, *font-size/line-height*, и *font-family* својства
@font-face	Правило које дозвољава да веб страна преузме и користи фонтове другачије од "*web-safe*" фонтова
font-family	Одређује *font family* за текст

font-feature-settings Дозвољава контролу над напредним типографским особинама у *OpenType* фонтовима
@font-feature-values Дозвољава ауторима да користе заједничка имена у *font-variant-alternate* за својства која се другачије активирају од *OpenType*
font-kerning Дозвољава коришћење *kerning* информације (размак између слова)
font-language-override Контролише коришћење *language-specific glyphs in a typeface*
font-size Одређује величину фонта у тексту
font-size-adjust Задржава читљивост текста када дође до пребацивање текста
font-stretch Бира нормалан, кондензован или проширеном облик из фамилије фонтова
font-style Одређује стил фонта за текст
font-synthesis Контрола за недостајући тип слова (подебљани или курзивни) који могу бити синтетизовани од стране претраживача
font-variant Одређује да ли ће текст бити приказан малим капитализованим фонтом
font-variant-alternates Контролише коришћење алтернативних глифова придружених алтернативним именима дефинисаним у @font-feature-values
font-variant-caps Контролише коришћење алтернативних знакова за велика слова
font-variant-east-asian Контролише коришћење алтернативних глифова за источноазијске скрипте (нпр. јапански и кинески)
font-variant-ligatures Контролише које лигатуре и контекстуалне форме се користе у текстуалном садржају елемената на које се односи
font-variant-numeric Контролише коришћење алтернативних знакова за бројеве, разломљене бројеве и редне бројеве-ознаке
font-variant-position Контролише коришћење алтернативних визуелно представљених слова и бројева (*glyphs*) мање величине, позиционираних као експонент или индекс у односу на основну линију фонта
font-weight Одређује тежински фактор фонта (колико се танко или дебело приказује фонт у тексту

grid Координатна мрежа или грид је скраћено својство за *grid-template-rows*, *grid-template-columns*, *grid-template-areas*, *grid-auto-rows*, *grid-auto-columns*, и *grid-auto-flow properties*
grid-area Специфицира име за ставку мреже или је ово својство скраћена особина за *grid-row-start*, *grid-column-start*, *grid-row-end*, и *grid-column-end* својство
grid-auto-columns Одређује подразумевану величину колоне
grid-auto-flow Одређује како се аутоматски постављене ставке убацују у мрежу
grid-auto-rows Одређује подразумевану величину за врсте
grid-column Скраћеница за својства *grid-column-start* и *grid-column-end properties*
grid-column-end Одређује где се завршава ставка у мрежи
grid-column-gap Одређује величину размака између колона
grid-column-start Одређује где почиње ставка у мрежи
grid-gap *A shorthand property for the grid-row-gap and grid-column-gap properties* Скраћеница за својства *grid-row-gap* и *grid-column-gap*
grid-row Скраћено својство за *grid-row-start* и *grid-row-end* својство
grid-row-end Одређује где се завршава ставка у мрежи

grid-row-gap	Одређује величину размака између редова
grid-row-start	Одређује где је почетак ставке у мрежи
grid-template	Скраћено својство за *grid-template-rows*, *grid-template-columns* и *grid-areas* особине
grid-template-areas	Одређује како се приказују колоне и редови, користећи именоване ставке мреже
grid-template-columns	Одређује величину колона и колико стубаца има у организацији мреже
grid-template-rows	Одређује величину редова у организацији мреже
hanging-punctuation	Одређује да ли се знак интерпункције може налазити изван линијског оквира
height	Поставља висину елемента
hyphens	Поставља како да се раздвоје речи ради побољшања изгледа параграфа
image-rendering	Даје инструкције прегледачу о томе које аспекте слике треба сачувати као најважније када се слика скалира
@import	Омогућава да се увезу стилови у други стил
isolation	Дефинише да ли елемент мора да креира нови садржај за слагање
justify-content	Одређује поравнање између ставки унутар флексибилног контејнера када ставке не користе сав расположиви простор
@keyframes	Одређује код анимације
left	Одређује леви положај позиционираног елемента
letter-spacing	Повећава или смањује размак између знакова у тексту
line-break	Одређује да ли треба и како да се прави прекид линије
line-height	Поставља висину линије
list-style	Поставља сва својства за листу у једној декларацији
list-style-image	Одређује слику као ознаку за *list-item* маркер
list-style-position	Одређује позицију *list-item* маркера (за означавања ставки листе које је уобичајено кружићима или тачкама)
list-style-type	Одређује тип *list-item* маркера
margin	Подешава сва својства маргине у једној декларацији
margin-bottom	Поставља доњу маргину елемента
margin-left	Подешава леву маргину елемента
margin-right	Поставља десну маргину елемента
margin-top	Поставља горњу маргину елемента
max-height	Поставља максималну висину елемента
max-width	Поставља максималну ширину елемента
@media	Поставља правила стила за различите врсте / уређаје / величине медија
min-height	Одређује минималну висину елемента
min-width	Поставља минималну ширину елемента
mix-blend-mode	Одређује како садржај елемента треба да се меша са његовом директном позадином претходника

object-fit	Одређује како садржај замењеног елемента треба да се постави у оквир који је утврђен његовом висином и ширином
object-position	Специфицира поравнање замењеног елемента унутар његовог оквира
opacity	Поставља ниво непрозирности за елемент
order	Поставља редослед флексибилне ставке релативно у односу на остатак
orphans	Поставља минимални број линија које се морају оставити на дну странице када дође до прелома унутар елемента
outline	Скраћено својство за приказ ширине, стила и својства боје
outline-color	Поставља боју приказа
outline-offset	Офсет приказа и извлачи га изван граничне ивице
outline-style	Подешава стил контуре - приказа
outline-width	Поставља ширину контуре - приказа
overflow	Одређује шта ће се десити ако садржај пређе преко оквира елемента
overflow-wrap	Одређује да ли или не прегледач може да прекида линије унутар речи, и како, да би спречио прекорачење (када је стринг предуг да би се уклопио у постојећи оквир)
overflow-x	Одређује да ли ће, или не, одсецати леви / десни део садржаја, ако прелази оквир у коме је садржај елемента
overflow-y	Одређује да ли да исеца, или не, горње / доње ивице садржаја, ако прелазе оквир садржаја елемента
padding	Скраћено својство за све *padding* особине (*padding* прави додатни празан простор у елементу или око елемента)
padding-bottom	Поставља доњи *padding* елемента
padding-left	Подешава леви *padding* елемента
padding-right	Поставља десни *padding* елемента
padding-top	Одређује горњи *padding* елемента
page-break-after	Поставља понашање прелома странице након елемента
page-break-before	Поставља понашање прелома странице пре елемента
page-break-inside	Поставља понашање прелома странице унутар елемента
perspective	Даје 3*D* позиционираним елементима неку перспективу
perspective-origin	Дефинише на којој позицији корисник гледа 3*D* позиционирани елемент
pointer-events	Дефинише да ли елемент реагује на догађаје показивача
position	Специфицира тип метода позиционирања који се користи за елемент (статички, релативни, апсолутни или фиксни)
quotes	Поставља врсту наводника за уграђене наводнике
resize	Дефинише да ли (и како) корисник може да промени величину елемент
right	Одређује праву позицију позиционираног елемента
scroll-behavior	Одређује да ли да континуално помера *scroll* позиција у *scroll* пољу, уместо скоковите промене (*scroll* је могућност да се помера приказ на екрану да би се видео нови материјал, на пример точкићем миша)
tab-size	Одређује ширину табулатор карактера

table-layout	Дефинише алгоритам који се користи за постављање ћелија табела, редова и колона
text-align	Одређује хоризонтално поравнање текста
text-align-last	Описује како се поравнава последња линија блока или линије тачно пре прекида линије када се поравнање текста постави на "*justify*"
text-combine-upright	Одређује комбинацију више знакова у простор једног карактера
text-decoration	Одређује декорацију која се додаје у текст
text-decoration-color	Одређује боју текста као декорацију текста
text-decoration-line	Одређује тип линије у декорацији текста
text-decoration-style	Одређује стил линије у декорацији текста
text-indent	Одређује увлачење прве линије у текстуалном блоку
text-justify	Одређује методу поравнања која се користи када је поравнавање текста "*justify*"
text-orientation	Дефинише оријентацију текста у линији
text-overflow	Одређује шта би требало да се деси када текст изађе изван простора за садржај елемента
text-shadow	Додаје сенку у текст
text-transform	Контролише капитализацију текста (мала слова се приказују као умањена велика слова)
text-underline-position	Одређује позицију доње линије која се поставља помоћу својства за декорацију текста
top	Одређује горњу позицију позиционираног елемента
transform	Примењује 2*D* или 3*D* трансформацију на елемент
transform-origin	Омогућава да се промени положај трансформисаних елемената
transform-style	Одређује како се угњеждени елементи приказују у 3*D* простору
transition	Скраћено својство за сва својства транзиције
transition-delay	Одређује када ће почети ефекат транзиције
transition-duration	Одређује за колико секунди или милисекунди треба да се заврши ефекат транзиције
transition-property	Специфицира име за *CSS* својство на које се односи ефекат транзиције
transition-timing-function	Одређује криву брзине транзиционог ефекта
unicode-bidi	Користи се заједно са својством управљања за постављање или враћање да ли текст треба да буде замењен да подржава више језика у истом документу
user-select	Одређује да ли се може изабрати текст елемента
vertical-align	Поставља вертикално поравнање елемента
visibility	Одређује да ли је или није видљив елемент
white-space	Одређује како се уређује бели простор унутар елемента
widows	Поставља минимални број линија које се морају оставити на врху странице када дође до прелома странице унутар елемента
width	Подешава ширину елемента
word-break	Одређује како речи треба да се преломе када стигну до краја линије
word-spacing	Повећава или смањује размак између речи у тексту
word-wrap	Омогућава прекидање дугих, нераскидивих речи и пребацивање у следећу линију

writing-mode	Одређује да ли се линије текста постављају хоризонтално или вертикално
z-index	Поставља редослед стека позиционираног елемента.

Више детаља може се наћи у уџбенику који је написао професор др Зоран Ћировић, Увод у Интернет технологије, ВИШЕР, 2015 [2015_Cirovic_UuIT].

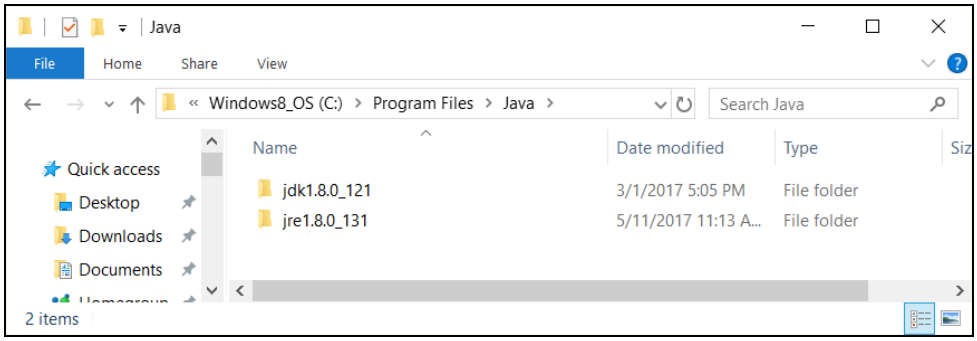
1.2.6 Јава програми

Већина корисника који користе неки од веб прегледача инсталирају Јава програм. Када се преузима Јава софтвер обично се мисли на *Java Runtime Environment* (*JRE*). *JRE* се састоји од Јава виртуелне машине (*Java Virtual Machine, JVM*), Јава основних класа (*Java platform core classes*) и одговарајућих софтверских библиотека (*Java platform libraries*). *JRE* је такозвани *runtime* део Јава софтвера. *JRE* је све што је потребно за исправан рад веб прегледача. *JRE* садржи компоненту која се назива *Java Plug-in* софтвер. Захваљујући овоме, *JRE* омогућава аплетима који су написани у Јава програмском језику да се извршавају у оквиру различитих прегледача. Важно је да се зна да *Java Plug-in* софтвер није програм који се посебно инсталира и који функционише независно од осталих програма.

Јава као програмски језик пише се стандаризованим језиком који не зависи од оперативног система инсталираног на рачунару. Коришћењем компајлера, од текстуалног Јава кода добијају се такозване класе које може да извршава виртуелна машина *JVM*, а сваки специфичан рачунар има одговарајући софтвер који ће ове класе претворити у команде које разуме оперативни систем. Због тога, они који пишу програме за веб апликације, не треба да размишљају о оперативном систему, јер би код требао да ради на сваком рачунару. *JVM* је само један аспект Јава софтвера који се користи у веб интеракцијама. Према томе, *JVM* је уграђена у Јава софтвер који је преузет са Интернета, и он помаже да се извршавају Јава апликације.

Власници Јаве софтвера (*Oracle*) стално раде побољшања да би унапредили перформансе, стабилност и сигурност рада, која је посебно важна за интернет апликације. Јава софтвер је бесплатан и може се слободно користити, па се препоручује да се инсталира најновија верзија или уради надоградња када се добије таква информација.

Међутим, када се спремамо да развијамо интернет апликације засноване на Јави, то није верзија која је комплетна. Потребно је да се инсталира развојно окружење *JDK* (*Java Development Kit*) која садржи Јава компајлер (*javac*). Инсталације ових програма су углавном на `C` диску у оквиру фолдера **`Program Files`**, а обе верзије (*JRE* и *JDK*) су у фолдеру **`Java`**. Слика 1.2.3 приказује где се налазе инсталације Јаве на рачунару наставника.



Слика 1.2.3 Фолдери у којима су инсталирани Јава програми *JRE* и *JDK*.

У даљем тексту се подразумева да студент има инсталиране *JRE* и *JDK*. Програмирање у Јава језику изучава се у посебном курсу.

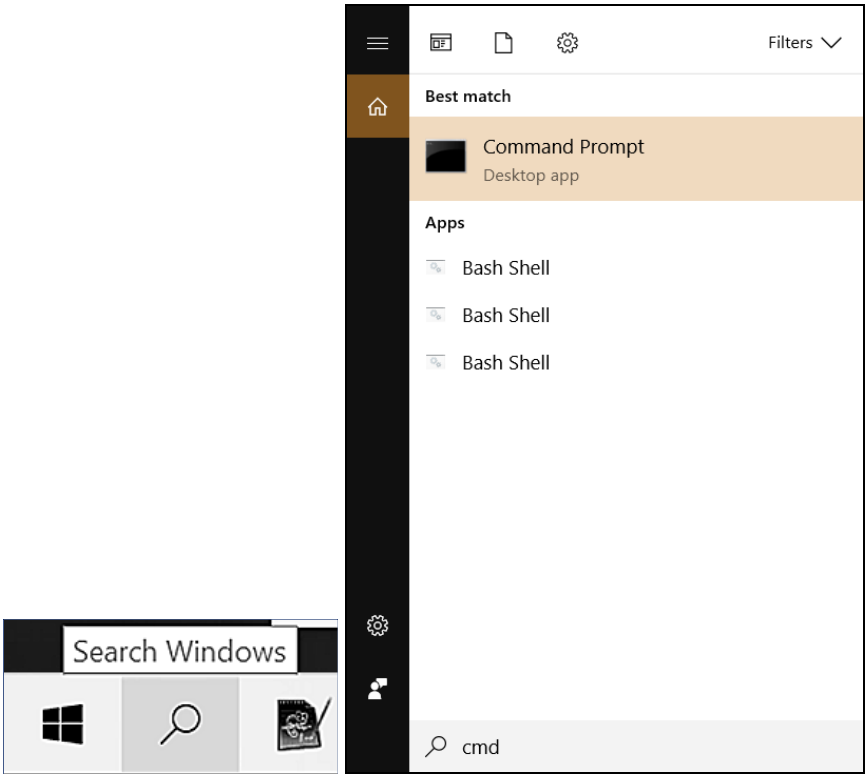
1.2.7 *CMD*, *Command Prompt*

За потребе аутоматизације и тестирања рада сервера, у овом курсу биће коришћен командни прозор, који се може наћи из менија са основног приказа рачунара.

На пример, када се у доњем левом углу прозора на рачунару на коме је инсталиран оперативни систем *Windows* 10, кликне на знак лупе, и откуца *CMD* или *Command Prompt*, појавиће се прозори као на Слици 1.2.4. Потребно је да се кликне на пронађену апликацију поред које је са леве стране црни правоугаоник.

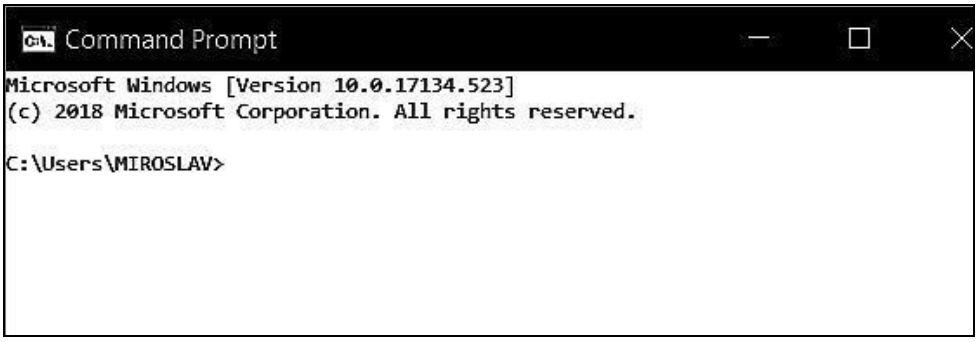
Након тога се отвара прозор са црном позадином као што је приказано на Слици 1.2.5. У врху прозора је текст који говори о томе који су оперативни систем рачунара, верзија и година.

```
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.
```



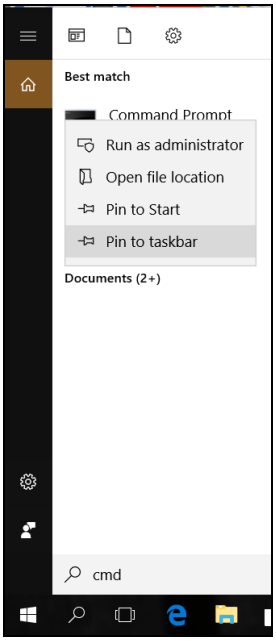
Слика 1.2.4 Налажење одговарајућег прозора за *CMD*.

Наредна линија овог прозора приказује у ком директоријуму се налаз курзор.



Слика 1.2.5 Почетни прозора отворен са *CMD* (инвертоване боје на слици).

```
C:\Users\MIROSLAV>
```



Слика 1.2.6 Постављање *CMD* у таскбар.

У том директоријуму рачунар очекује да се унесе команда или инструкције шта треба рачунар да уради. Рачунар има унапред дефинисане путање где би могле да се нађу команде или називи извршних програма. Ако желимо да извршимо неки програм, а не налази се у подразумеваном директоријуму, та команда или програм се неће извршити.

Касније ће се појавити потреба да се *CMD* прозор често отвара, па је погодно да се одмах постави иконица екрана са црном позадином у таскбар, који се најчешће налази у дну десктопа рачунара, или се појављује када се мишем приближи у део екрана који садржи таскбар. На Слици 1.2.6 се види део таскбара где су иконице за проналажење програма (знак лупе), Интернет експлорер (знак малог слова е), знак фасцикле (где се отвара неки директоријум).

Када се добије приказ као на Слици 1.2.6, кликом на *Pin to taskbar*, црна икона ће се појавити у таскбару (као да смо чиодом причврстили папир на дрвену таблу са подсетницима).

Када се касније укаже потреба да преко *CMD* прозора унесемо захтев или команду, довољно је да се једном кликне на црну иконицу на таскбару.

1.2.8 Питања и задаци за вежбу

1. Пронаћи Команд промпт и поставити га у таскбар.
2. Пронаћи где је радни директоријум на рачунару.
3. Ископирати *html* фајл са интернета или неки постојећи *html* фајл са рачунара у радни директоријум.

2 Инсталација сервера

Резиме

Циљ овог поглавља је да се студент оспособи за инсталацију веб сервера и оперативан рад са сервером како би на свом рачунару могао да направи развојно окружење и тестира спликације.

2.1 Инсталација *Apache Tomcat* сервера

Резиме

Циљ овог поглавља је да студент оспособи да преузме инсталационе фајлове за сервер и да их исправно инсталира на свом рачунару. Цео поступак је објашњен корак по корак. Дати су најосновнији појмови који укључују основне појмове о *Apache Tomcat HTTP* серверу, Инсталација *Apache Tomcat* која се састоји од преузимање инсталационе датотеке и инсталација, прављење директоријума у Виндоус окружењу и са *Command Prompt*, опис садржаја *Tomcat* фолдера, прављење променљиву `"JAVA_HOME"` у Виндоус окружењу и прављење променљиве `"JAVA_HOME"` са *Command Prompt*.

Увод у инсталацију Apache Tomcat сервера

Овај део поглавља садржи инструкције за инсталацију *Apache Tomcat* сервера на рачунару клијента. Наиме, у овом курсу треба омогућити програмирање интернет апликација тако да програмер има на истом рачунару и клијентски део и серверски део. На овај начин програмер има потпуну контролу на страни и клијентског и серверског рачунара, и може да коригује програме одмах када уочи грешку или програм не даје резултате како је програмер осмислио да све треба да функционише. Када је програмер задовољан урађеном апликацијом, он треба да закупи простор на неком серверу, да инсталира серверски софтвер на серверу, да копира све фајлове у одговарајуће директоријуме на серверу, да уради модификације фајлова ради спречавања злоупотреба и стабилног и сигурног рада, и на крају да тестира рад тако што би са удаљеног клијентског рачунара приступио одговарајућем садржају на серверу, и верификовао да апликација исправно ради.

За успешну реализацију ове теме, потребно је добро познавање клијент-сервер архитектуре што је обрађено у претходним поглављима.

У другом блоку овог курса биће урађени примери на бази Сервлет технологија. За разлику од статичких веб страница које могу да се постављају на серверу, сервлети ће бити коришћени за генерисање динамичких веб страница на серверској страни. Сматра се да је Сервлет технологија робусна и скалабилна зато што је заснована на коришћењу Јава језика. Постоји много интерфејса и класа у *API* сервлетима (*Application Programming Interface*) као што су *Servlet, GenericServlet, HttpServlet, ServletRekuest, ServletResponse*.

2.1.1 *Apache Tomcat HTTP* сервер

Појмови који ће бити коришћени у овом поглављу су *HTTP* Сервер, *HTTP Client* (веб прегледач), *Database* (база података), *Client-Side* програми, *Server-Side* програми. *Tomcat* је *HTTP* апликација која ради преко *TCP/IP*. *HTTP* протокол апликацијског слоја се покреће преко *TCP/IP*; *IP* за усмеравање и адресирање јединственом *IP* адресом за рачунаре на Интернету). *TCP* подржава мултиплексирање преко софтверских портова.

Tomcat сервер ради на специфичним *TCP* портовима са специфичном *IP* адресом; подразумевани број *TCP* порта за *HTTP* протокол је 80. За тестирање *HTTP* сервера, може се изабрати било који број неискоришћених портова између 1024 и 65535.

HTTP је асинхрони протокол: клијент шаље захтев серверу; сервер враћа клијенту поруку са одговором (синтакса поруке је у *HTTP* спецификацији).

Apache Tomcat је Јава-орјентисан *HTTP* сервер који може да извршава специјалне Јава програме, "*Java Servlet*" и "*Java Server Pages*" (*JSP*).

Производ је отвореног типа, под лиценцом "*Apache Software Foundation*".

За инсталацију *Apache Tomcat* потребно је да се оде на неки од њихових сајтова

https://www.apache.org/,

https://httpd.apache.org/download.cgi,

http://tomcat.apache.org/

а тренутно коришћена верзија је *Apache Tomcat* 9.x, која се користи за *Servlet* 4.0, *JSP* 2.3, *EL* 3.0, *WebSocket* 1.1 и *JASPIC* 1.1.

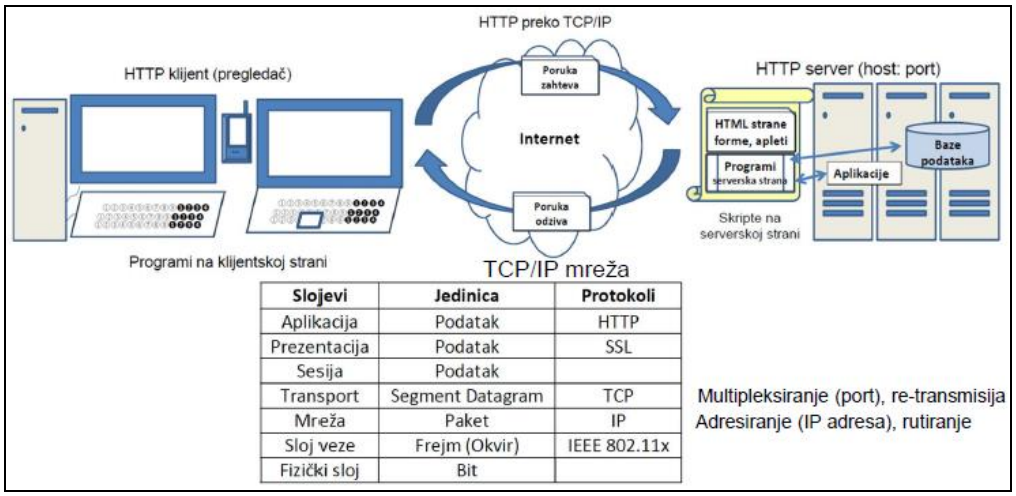
Свака новија верзија има нове могућности, а могу се инсталирати следеће верзије:

- Tomcat 3.x: RI for Servlet 2.2 and JSP 1.1
- Tomcat 4.x: RI for Servlet 2.3 and JSP 1.2
- Tomcat 5.x: RI for Servlet 2.4 and JSP 2.0
- Tomcat 6.x: RI for Servlet 2.5 and JSP 2.1
- Tomcat 7.x: RI for Servlet 3.0, JSP 2.2 and EL 2.2
- Tomcat 8.x: RI for Servlet 3.1, JSP 2.3, EL 3.0, & Java WebSocket 1.0

За потребе овог курса коришћена је верзија Tomcat 8.x (добро је користити предзадњу објављену верзију јер је она обично стабилна, а свака нова верзија пролази кроз модификације и такозване закрпе уочених недостатака).

Најпознатији *HTTP* сервери су*: Apache HTTP Server, Apache Tomcat Server, Microsoft Internet Information Server, Google Web Server*.

Најчешће коришћени *HTTP* клијенти (веб прегледач) су: *Chrome, FireFox, Internet Explorer*.



Slojevi	Jedinica	Protokoli
Aplikacija	Podatak	HTTP
Prezentacija	Podatak	SSL
Sesija	Podatak	
Transport	Segment Datagram	TCP
Mreža	Paket	IP
Sloj veze	Frejm (Okvir)	IEEE 802.11x
Fizički sloj	Bit	

Слика 2.1.1 Пример *HTML* клијената и *HTML* сервера.

Као база података (*Database*) могу се користити: *MySQL, Apache Derby, mSQL, SQLite*.

На страни клијента програми могу бити написани у *HTML, JavaScript, VBScript, Flash*.

На страни сервера програми могу бити написани у *Java Servlet/JSP, ASP, PHP, Python*.

На слици 2.1.1 је нацртан пример клијент-сервер архитектуре.

2.1.2 Инсталација *Apache Tomcat*

Инсталирање сервера на клијентском рачунару биће објашњена кроз неколико корака са додатним објашњењима. Инсталирање сервера на истом рачунару на коме је и клијент има сврху да програмер који развија своју динамичку апликацију може одмах да провери да ли је успешно написао програме пре него што их пребаци на сервер.

2.1.2.1 Корак 1, преузимање инсталационе датотеке и инсталација

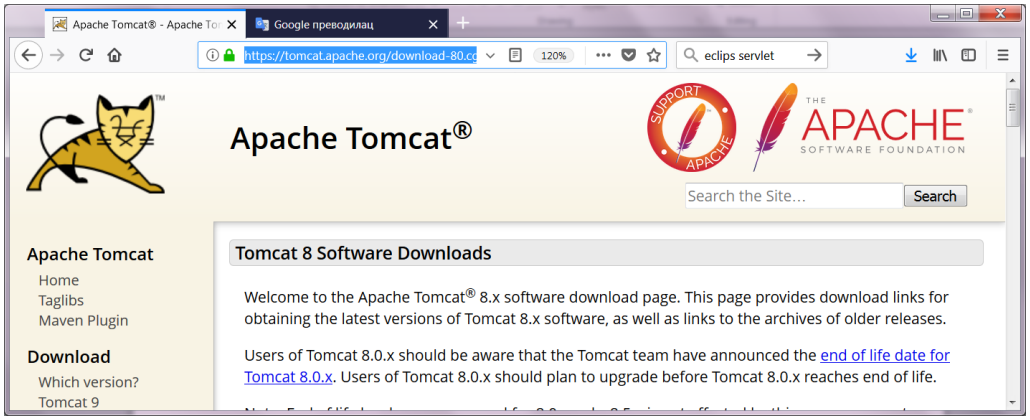
Преузети инсталацију са сајта *Apache Tomcat*, https://tomcat.apache.org/download-80.cgi

На слици 2.1.2 је изглед веб странице у време писања овог поглавља, а како је и коришћено у курсу 2019/2020. Често се дешава да се после писања књиге појаве нове верзије зато што осим писања, уџбеник морају да оцене и рецензенти, да се уради техничка обрада, и да се одобри публиковање од стране наставно-стручног већа. С друге

стране, студенти могу да користе и неку претходну верзију уместо најновије, као на пример ону која је коришћења у овом уџбенику, ако неки од корака са новијим верзијама не дају очекиване резултате.

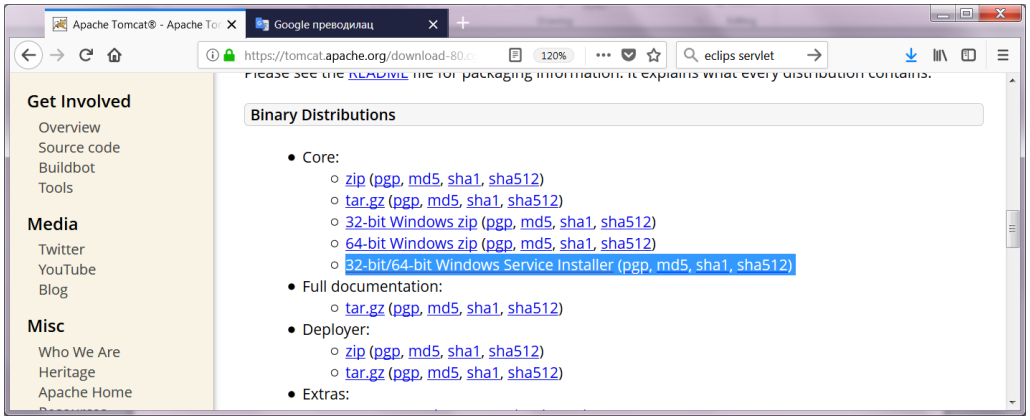
На сајту http://tomcat.apache.org, скролује се страница док се не дође до текста "Tomcat 8.5.{xx} Released" (где је {xx} број најновије верзије). Кликне се мишем на Downloads који је на дну параграфа. Долази се на страницу која садржи све додатне информације о преузимању датотеке за инсталирање сервера, уз пратеће фајлове који се касније могу користити као упутство за коришћење. Верзија која је подржана у време писања је "8.5.{xx}". Препоручујем да изаберете *Binary Distributions*, *Core*, и то архивирану датотеку "ZIP" package (на пример "apache-tomcat-8.5.30"; у овом поглављу је коришћена верзија "apache-tomcat-8.5.24.zip", која је величине око 10 MB) за *Windows*, за 64-бита. За производну верзију треба користити инсталациону верзију.

За обуку и тест, боље је да се користи бинарна верзија која се не инсталира, већ само копира и распакује ако је преузета као архивирана датотека. На слици 2.1.3 је приказана страница са које је преузета архивирана бинарна верзија софтвера. Уколико клијентски рачунар ради на 32 бита, тада треба преузети одговарајућу верзију, као што је показано на слици 2.3.



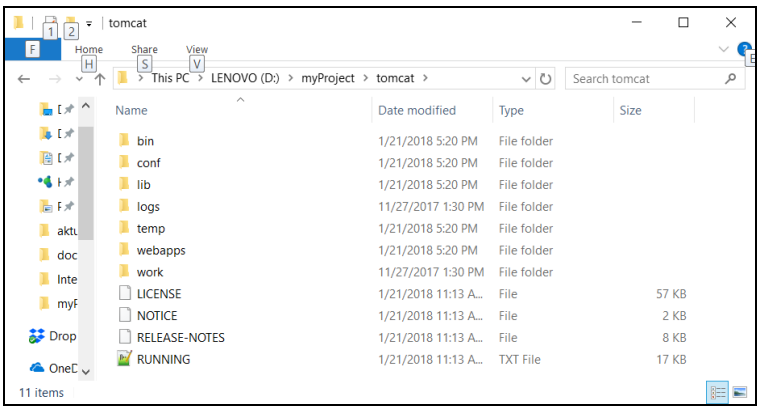
Слика 2.1.2 Изглед веб странице за преузимање инсталације *Apache Tomcat* сервера.

Пре распакивања преузете архиване датотеке, најбоље да се направи директоријум у који ће бити смештени фајлови сервера. Будућу да ће бити потребно да често радимо неке операције, а да не би имена фајлова са путањама до њих била превише велике дужине, најбоље да директоријум буде са кратким и лако приступачним називом. Ако постоје два диска (не мора физички два диска, већ две партиције на истом диску, на пример *C*: и *D*:), уобичајено се *C*: партиција користи за инсталирање програма а *D*: партиција за радне фајлове. Изабраћемо назив директоријума на *D*: партицији, на пример "*D*:\myProject". Уместо великог слова "*D*:" може се користити и мало слово "*d*:". Треба направити пројектни директоријум, "d:\myProject", и у њему распаковати *zip* архиву "apache-tomcat-8.5.24.zip", и добија се фолдер (користи се термин фолдер који значи исто што и директоријум) "d:\myProject\apache-tomcat-8.0.24"



Слика 2.1.3 Страница са које је преузета верзија *Apache Tomcat* сервера.

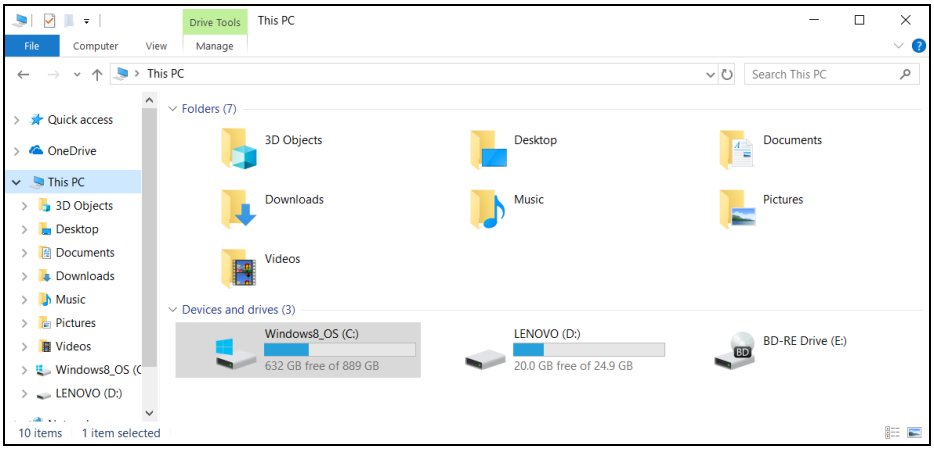
Ради лакше употребе, коришћења краћих назива фајлова и директоријума преименовати фолдер "apache-tomcat-8.0.24" у "tomcat" Сада се каже да је *Tomcat installed directory* "d:\myProject\tomcat"; увек када се у тексту касније каже да треба да се уради нешто у *Tomcat* инсталационом директоријуму, мисли се на "d:\myProject\tomcat". Инсталациони директоријум ће у подешавању добити и своје симболичко име које ће референцирати као <TOMCAT_HOME>. На слици 2.1.4 је приказ садржаја инсталационог *Tomcat* директоријума.



Слика 2.1.4 Садржај инсталационог директоријума *Tomcat* сервера.

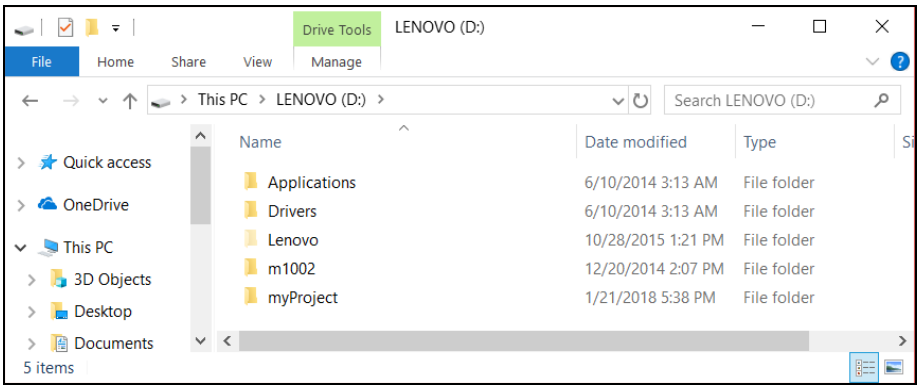
2.1.2.2 Прављење директоријума у Виндоус окружењу

За оне студенте који не знају како да направе нови директоријум, биће поновљене основне операције. Може се отворити било који фолдер, а затим у левој колони кликнути на *This PC*. Добија се нешто као на Слици 2.1.5, где препознајемо који хард дискови или партиције постоје у рачунару. Ако постоји диск или партиција "*D*:", двоструким кликом на ту сличицу се долази у корен (енглески термин је *root*) диска, као што је показано на слици 2.1.6.



Слика 2.1.5 Садржај директоријума једног рачунара са Виндоус оперативним системом.

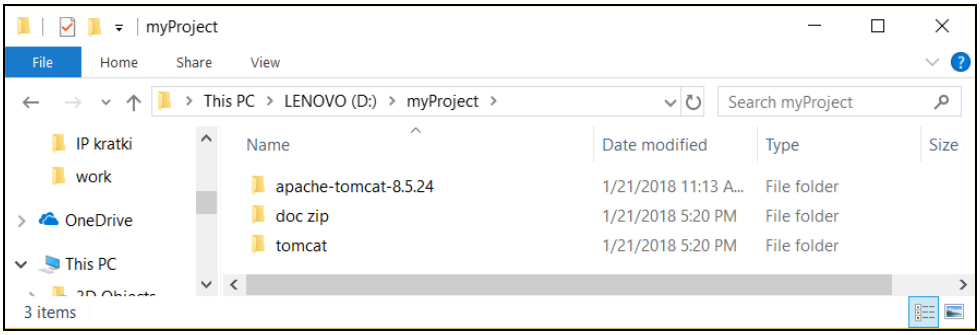
Десним кликом мишем и избором *New*, и уношењем назива фолдер направљен је нови фолдер на диску. Десним кликом на фолдер може се променити назив фолдера, или ископирати фолдер или обрисати фолдер. Када се направи фолдер "myProject", у њега се пребаци *zip* архиву "apache-tomcat-8.5.24.zip", а затим се распакује коришћењем одговарајућег софтвера. Уколико нема инсталираног софтвера за распакивање *zip* архиве, може се преузети са интернета и инсталирати програм који се зове 7-Zip; то је бесплатан програм отвореног кода за архивирање и распакивање групе фајлова заједно са фолдерима. Уколико је име архивиране датотеке превише дугачко, као што је у овом примеру случај, "apache-tomcat-8.5.24.zip", тада би и назив фолдера био дугачак, "apache-tomcat-8.5.24". Може се одмах променити име фолдера у неко краће име, на пример у "tomcat", али је боље да се задржи оригинални назив "apache-tomcat-8.5.24" и направи копија, којој ће назив бити "tomcat".



Слика 2.1.6 Садржај директоријума "*D*:" диска.

Задржавање оригиналне верзије распаковане архиве и копије може бити важно зато што у радној верзији треба променити неке вредности у оригиналним фајловима. Уколико

грешком променимо неки параметар у фајловима у фолдеру "tomcat", увек можемо ископирати оригинални фајл из фолдера "apache-tomcat-8.5.24" и урадити измене од почетка.

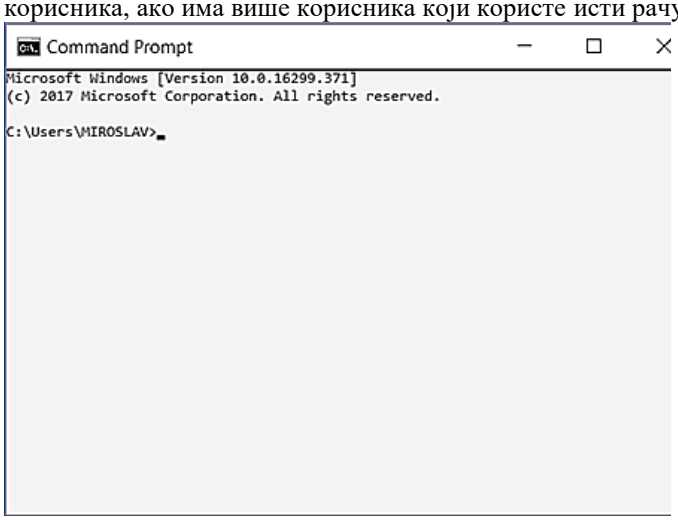


Слика 2.1.7 Садржај директоријума "myProject".

На слици 2.1.7 је садржај директоријума у који ће бити фајлови серверске стране. Радни директоријум "tomcat" служи да се у њему модификују фајлови током развоја апликација. Фолдер "apache-tomcat-8.5.24" садржи оригиналне фајлове пре модификације. Фолдер "doc zip" садржи различите верзије архива, као и документацију и упутства.

2.1.2.3 Прављење директоријума са *Command Prompt*

Прављење фолдера, брисање, копирање фајлова и фолдера може да се уради и коришћењем *CMD*. Кликом на црну иконицу у таскбару на дну десктопа, отвара се радни виндоус прозор, као на слици 2.1.8. У примеру са слике појављује се да је радни директоријум `C:\Users\MIROSLAV>`, `Microsoft Windows Version 10, (c) 2017`. Код неких рачунара се уместо `Users` може појавити `Korisnik`. Уместо `MIROSLAV` појавиће се име корисника, ако има више корисника који користе исти рачунар.

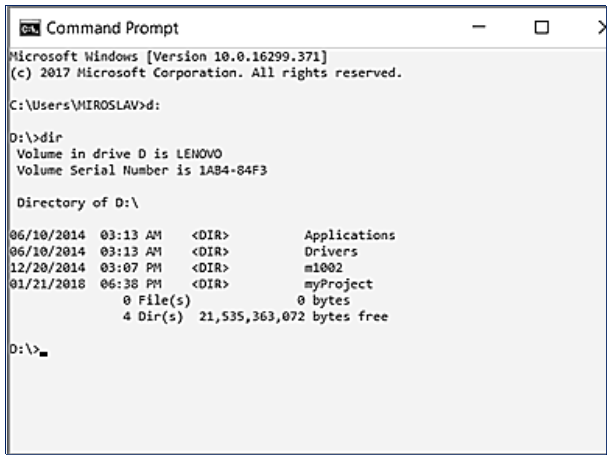


Слика 2.1.8 Стартовање *Command Prompt* иконице на рачунару.

Прелазак на "D:\>" партицију се остварује уношењем `d:`, као што је показано на слици 2.1.9.

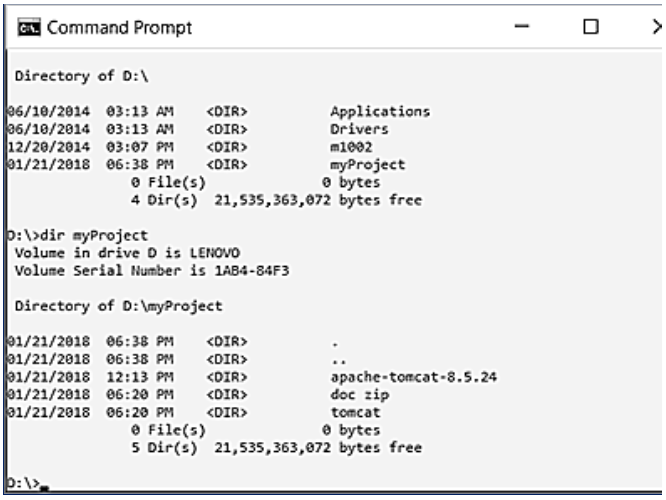
Текстуални приказ фајлова који се налазе у неком директоријуму се добија са "dir", као што је показано на слици 2.1.10.

Слика 2.1.9 Прелазак на “D:\>” партицију.



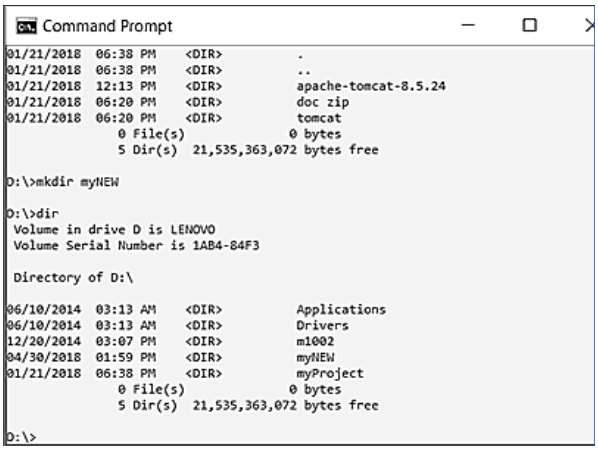
Слика 2.1.10 Приказ фајлова и директоријума у текућем директоријуму.

Текстуални приказ фајлова који се налазе у неком директоријуму испод текућег, добија се са "dir myProject", као што је показано на слици 2.1.11. У суштини, ако се откуца у линији пун назив директоријума одмах после "dir ", добија се садржај тог директоријума.



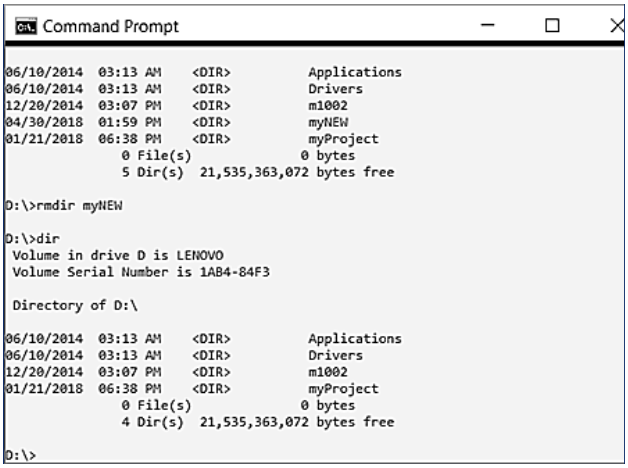
Слика 2.1.11 Приказ фајлова и поддиректоријуму "myProject".

На слици 2.1.12 је показано да се командом "mkdir myNEW", прави нови директоријум "myNEW" у текућем, а затим приказује садржај директоријума да би се проверило да ли је команда извршена и стварно направљен нови директоријум "myNEW".



Слика 2.1.12 Прављење новог директоријума "myNEW" и приказ текућег директоријума.

На слици 2.1.13 је показано да се командом "rmdir myNEW" брише директоријум "myNEW" у текућем, а затим приказује садржај директоријума да би се проверило да ли је команда извршена и стварно обрисан директоријум "myNEW".



Слика 2.1.13 Брисање директоријума "myNEW" и приказ текућег директоријума.

Куцањем текста "exit" излази се из команд промпта и затвара прозор. Ако је направљен нови директоријум, он ће остати ту где је направљен и биће видљив и из графичког окружења.

Друга важна карактеристика која се може уочити јесте да се команде уносе линија по линија и одмах интерпретирају, одакле и потиче назив овакве врсте рада. Већини обичних корисника рачунара овакав начин рада може да буде превише компликован и непотребан. Програмери који се бави Интернет програмирањем би требало да знају за овакав начин рада, и да га користе, зато што рачунар у истом прозору враћа информације шта је урадио, шта је добро а шта су грешке, и то линија по линија одговора, а што је најчешће сакривено од корисника или корисници морају да покрећу нове програме за мониторинг рада апликација.

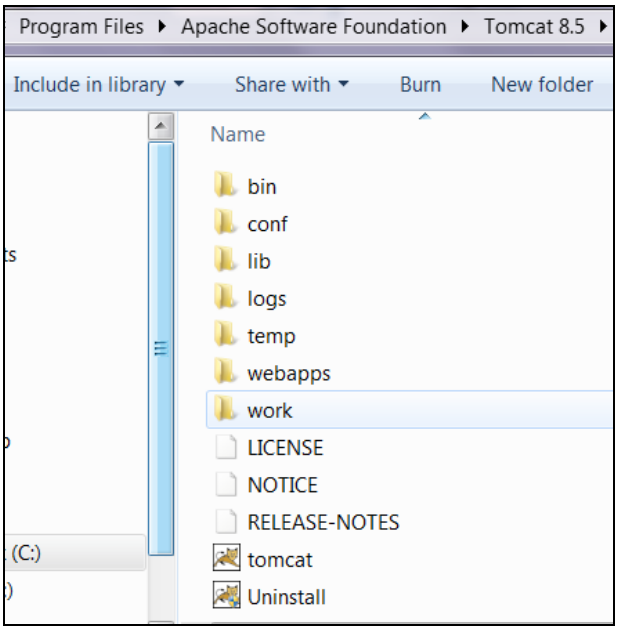
Иако на први поглед може да изгледа да је непотребно да се команде уносе са тастатуре, једна или више линија кода се могу сместити у текстуалне фајлове са тачно одређеном екстензијом, па се чак и веома комплексан сет инструкција може извршити куцањем назива фајла. Такође је важно уочити да се куцањем текста "exit" излази из ове врсте рада, а на сличан начин се стартује или прекида рад сервера.

2.1.2.4 Садржај *Tomcat* фолдера

На слици 2.1.14 приказан је садржај *Tomcat* фолдера

- *bin*: садржи бинарне и скрипт фајлове, *startup script* (*startup.bat for Windows*), *shutdown script* (*shutdown.bat for Windows*)
- *conf*: садржи конфигурационе фајлове; *server.xml, web.xml, context.xml, tomcat-users.xml*
- *lib*: садржи *Tomcat system JAR files*, којима приступају све *vebap*; могу да буду и екстерни *JAR* фајлови (*MySQL JDBC Driver*)
- *logs*: *Tomcat log* фајлови; *error* поруке

- *webapps*: садржи *vebap* које ће се развијати; *Webapp Archive*
- *work*: *Tomcat* радни фолдер који користе *JSP*, *JSP-to-Servlet* конверзија
- *temp*: привремени фајлови

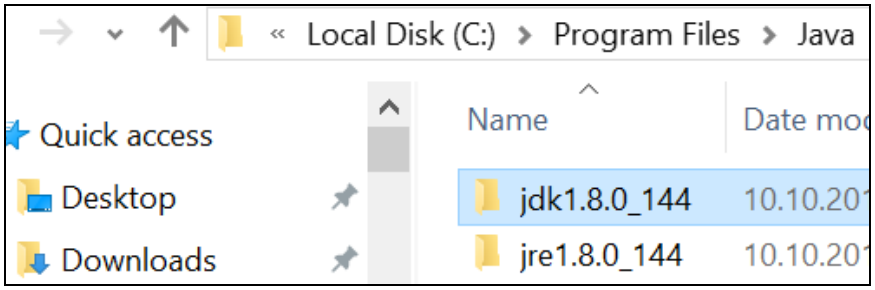


Слика 2.1.14 Садржај *Tomcat* фолдера.

2.1.2.5 Корак 2, прављење *JAVA_HOME* променљиве у окружењу

У другом кораку је потребно да се подеси системска променљиве у окружењу, *JAVA_HOME*, која треба да указује где се налази инсталација Јаве без обзира на верзију која се користи. Потребно је направити променљиву *environment variable* која се зове "JAVA_HOME" и поставити је да указује на *JDK* инсталациони директоријум.

Прво треба да се пронађе где је директоријум у који је инсталиран *JDK*; на пример "c:\Program Files\Java\jdk1.8.0_{xx}", где је {xx} број надградње, на пример са слике 2.1.15 то је 144, а фолдер је `c:\Program Files\Java\jdk1.8.0_144`.



Слика 2.1.15 Проналажење где је инсталиран *JDK* и који је број верзије (144).

2.1.2.6 Прављење променљиву "JAVA_HOME" у Виндоус окружењу

Да би направили променљиву "JAVA_HOME" у *Windows* 7/8/10, треба урадити следећим редом:

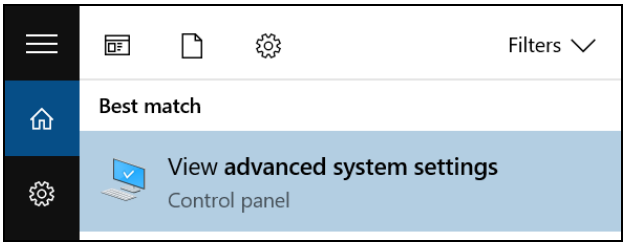
У *Start* "Control Panel" изабрати
System and Security (*Optional*)
System
Advanced system settings
Switch to "Advanced" tab
Environment Variables
System Variables
"New"
у "Variable Name" унети "JAVA_HOME"
у "Variable Value" унети *JDK* инсталирани директоријум из првог корака, на пример

`c:\Program Files\Java\jdk1.8.0_144`.

Алтернативно, преко *Search* знака који личи на лупу (као на слици 2.1.16, и куцањем у празном пољу текст *Advanced system settings*) пронаћи где се подешава.

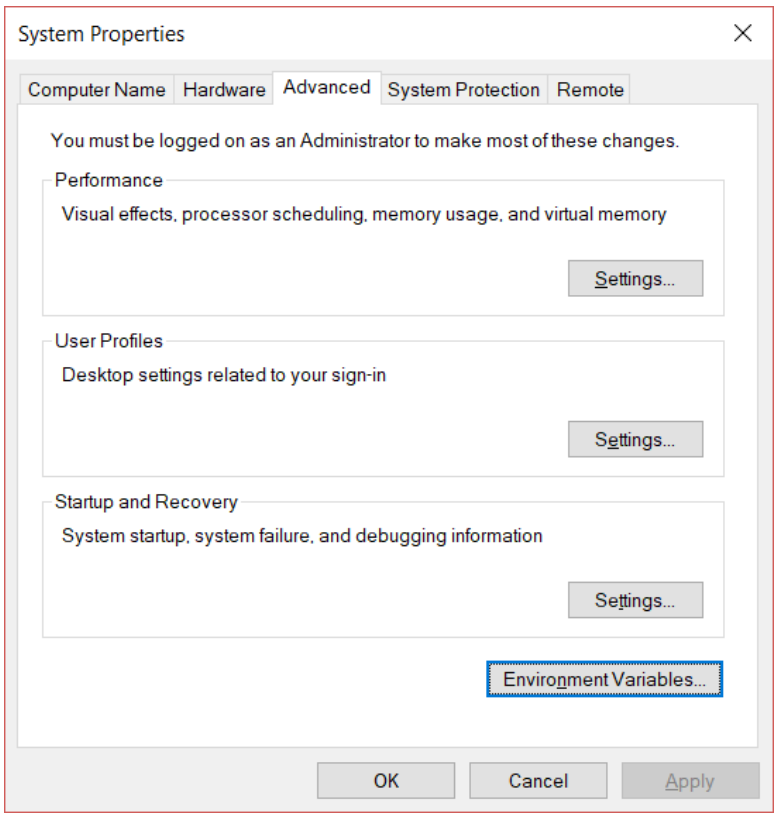
Слика 2.1.16 Коришћење *Search* (знак који личи на лупу, и куцање текста *Advanced system settings*).

На слици 2.1.17 је пронађено да се ова подешавања раде у *Control panel*, па се кликом мишем на *View*, долази до панела за промену сетованих параметара.



Слика 2.1.17 Коришћење *Search* и проналажење где је *Advanced system settings.*

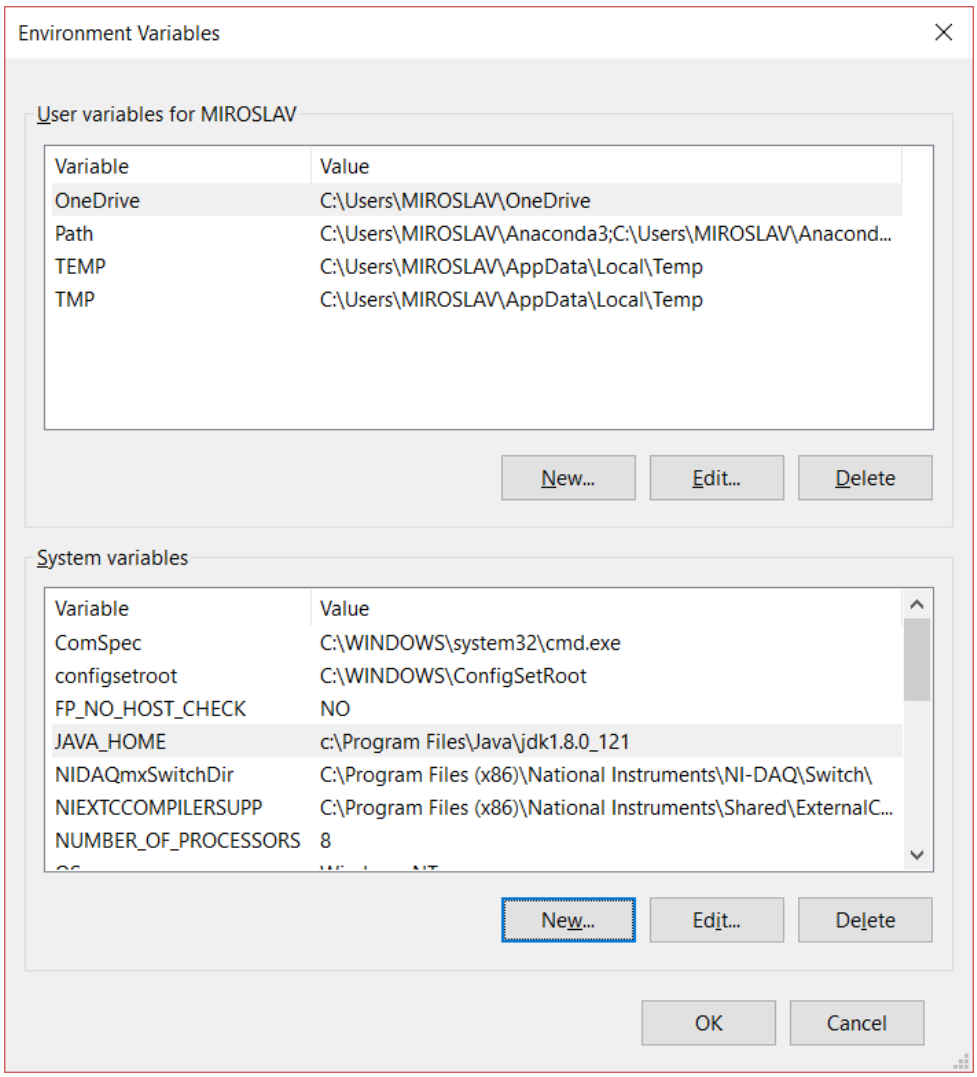
На слици 2.1.18 се види да се дошло до *System Properties*, одакле се кликом мишем на Environment Variables долази до панела као на слици 2.1.19, где се види да нема променљиве "JAVA_HOME".



Слика 2.1.18 Коришћење *Search* (знак који личи на лупу, и куцање текста *Advanced system settings*).

Ако нема променљиве, треба је додати кликом на дугме "NEW ", а као вредност унети

c:\Program Files\Java\jdk1.8.0_144. (На слици је илустрација за раније инсталиран *JDK* који је имао број 122, после инсталације нове верзије *JDK* треба да буде 144).



Слика 2.19 преко *Search* и куцањем текста *Advanced system settings.*

Пример подешавања параметара у Виндоус системима показује да оваква подешавања постају превише компликована за обичног корисника. Разлог за компликовање је вероватно тај да корисници не би могли лако да пронађу оваква подешавања, и највероватније направила грешку коју је тешко открити. Због тога нове верзије оперативних система постају озбиљна препрека програмерима који немају искуство са

оваквим подешавањима. Међутим, они који желе да се баве Интернет програмирањем треба да знају даураде и оваква подешавања.

2.1.2.7 Прављење променљиве "JAVA_HOME" са *Command Prompt*

Нека подешавања је једноставније да сеураде са *Command Prompt*. Већ смо у овом поглављу сугерисали да се у таскбару извуче иконица за *Command Prompt* односно *CMD*. Без обзира у ком директоријуму се налазите, довољно је да се откуца:

```
SET JAVA_HOME
JAVA_HOME=c:\Program Files\Java\jdk1.8.0_121
```

Променљива је подешена. Каже се да се стартује *CMD shell*.

Очигледно је да је познавање оперативног система и издавање команди из командне линије или подешавање параметара једноставније коришћењем *CMD*.

2.1.3 Питања и задаци за вежбу

1. Пронаћи *Apache Tomcat* и преузети инсталациони фајл који одговара оперативном систему и карактеристикама процесора.
2. Направити фолдер у који ће бити инсталиран *Tomcat* сервер.
3. Проверити да ли је инсталиран *JDK*. Ако није инсталиран *JDK*, преузети са Интернета инсталациони фајл, и инсталирати *JDK*.
4. Инсталирати *Tomcat* сервер и подесити "JAVA_HOME".

2.2 Конфигурисање *Apache Tomcat* сервера

Резиме

Циљ овог поглавља је да се студент оспособи да исправно конфигурише *Apache Tomcat HTTP* сервер. Детаљно је објашњен поступак конфигурисање *Apache Tomcat* сервера кроз опис конфигурације *Tomcat* сервера који има кораке *Set TCP Port Number* у `"conf\server.xml"`, *Enabling Directory Listing* у `"conf\web.xml"`, *Enabling Automatic Reload* у `"conf\context.xml"`, опциону измену у `"tomcat-users.xml"`, покретање и заустављање рада сервера. Дати су и детаљи за садржај покренутог *Tomcat* сервера, покретање прегледача као *HTTP* клијента да приступи серверу, заустављање сервера, поновно покретање сервера. Приказана је и статичка страна на серверу као провера да је конфигурација успешно урађена.

Увод у конфигурисање *Apache Tomcat* сервера

Ово поглавље садржи други део инструкција за инсталацију *Apache Tomcat* сервера на рачунару клијента које се односи на конфигурисање сервера. Програмер развија програме који треба да буду на серверском рачунару, али за потребе развоја алгоритма, тестирање апликације, верификацију и отклањање грешака, може да користи клијентски рачунар пре него што пребаци све фајлове на серверски рачунар. У оквиру претходног поглавља обрађено је како се преузимају инсталациони фајлови са Интернета и инсталира серверски

део софтвера на клијентском рачунару, као и подешавање једног системског параметра, кроз поглавља која чине корак 1 и корак 2. Ово поглавље наставља са следећим корацима инсталације сервера.

2.2.1 Конфигурисање *Apache Tomcat HTTP* сервер

Apache Tomcat HTTP сервер који се обрађује у овом курсу намењен је за рачунаре са Виндоус оперативним системом (*Windows operating system*) а за који се користи Јава програмски језик због независности од хардверске платформе на којој се развија интернет апликација и скалабилности. Програмер треба да се оспособи да кроз примере разуме израду апликација за једноставније потребе, а касније да проширује функционалност на сложеније случајеве.

У кораку 1 је описано преузимање инсталационе датотеке са Интернета и инсталација сервера на клијентском рачунару. У кораку 2 је направљена JAVA_HOME променљива која је постављена као системска варијабла.

2.2.1.1 Корак 3, конфигурација *Tomcat* сервера

Ако се инсталира бинарна верзија, тада је инсталациони директоријум са кратким називом путање ("myProject\tomcat\"), на пример на партицији диска где су програми корисника, "d:", са директоријумом који је одмах испод кореног директоријума, "d:\myProject\tomcat\"

Да би се извршила подешавања сервера, потребно је урадити измене у конфигурационим фајловима, који се налазе у директоријуму

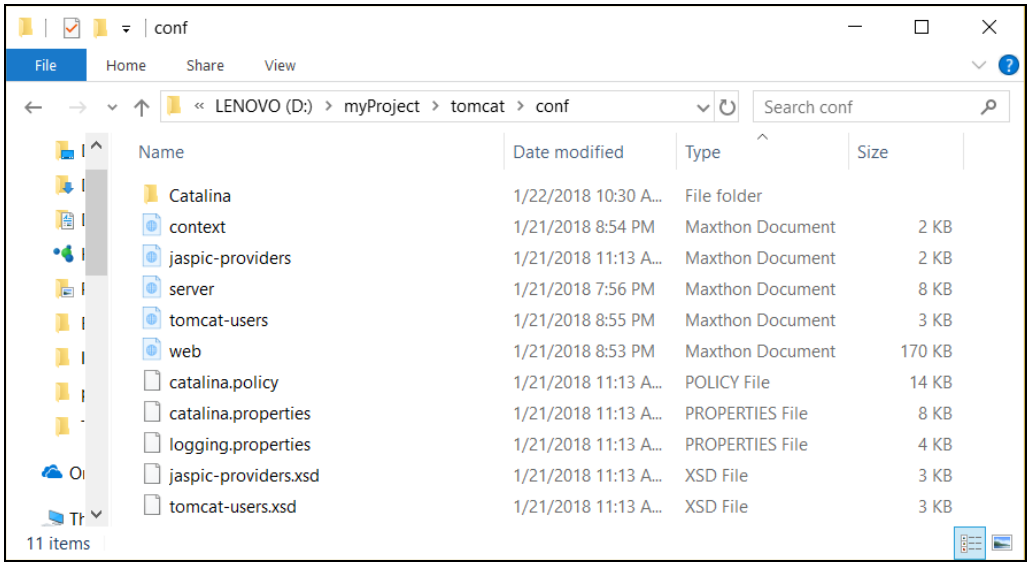
"d:\myProject\tomcat\conf "

На слици 2.2.1 је приказан садржај "...\tomcat\conf "директоријума.

Ради брзог отклањања погрешних корекција у конфигурационим фајловима, пожељно је да постоји резервна копија свих оригиналних фајлова који се налазе на пример у

"d:\myProject\apache-tomcat-8.5.24 ", при чему су конфигурациони фајлови у

"d:\myProject\apache-tomcat-8.5.24\conf "



Слика 2.2.1 Садржај *Tomcat* директоријума *conf*.

На овај начин знамо и која је верзија сервера инсталирана и увек може лако да се утврди који су фајлови промењени на основу датума модификације фајла.

Фајлови који ће бити модификовани су следећи:

1. server.xml
2. web.xml
3. context.xml
4. tomcat-users.xml

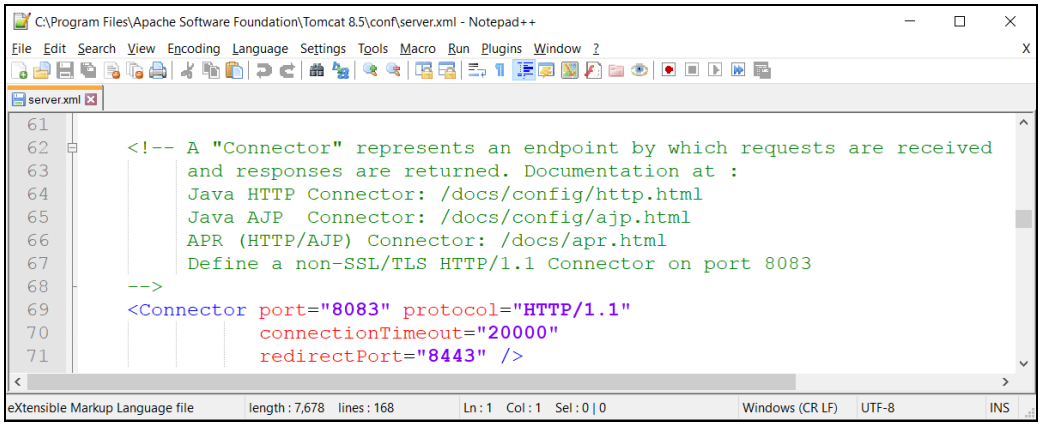
Следи опис промена које треба урадити у овим фајловима. За измене треба користити неки од текстуалних едитора типа Ноутпед (на пример *NotePad++*), да се фајловима не би пренели неки скривени невидљиви карактери које обично праве едитори текста вишег нивоа као што је Ворд.

2.2.1.2 *Set TCP Port Number* у "conf\server.xml"

На слици 2.2.2 је приказана део фајла "server.xml" у који треба да се ураде измене.

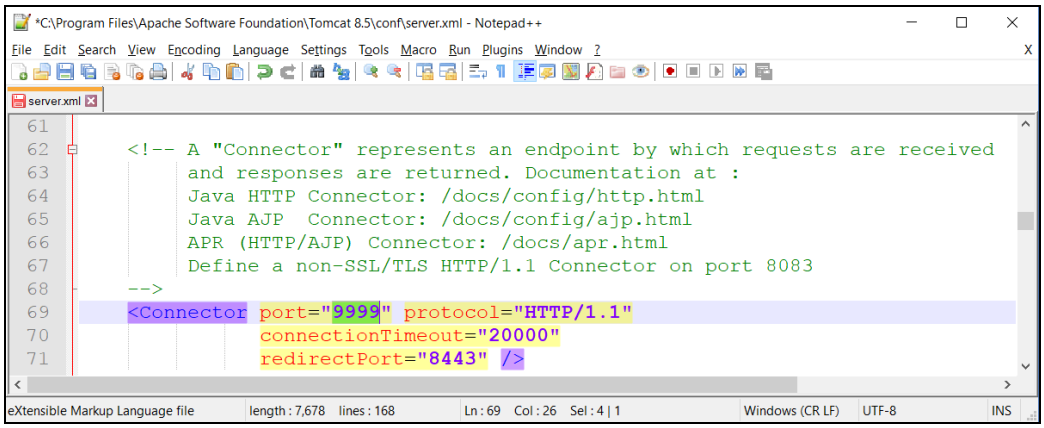
Подразумевани број *TCP* порта је 8080, уместо кога може да се изабере било који број из опсега 1024 - 65535, ако није већ предвиђено да нека друга постојећа апликација не користи неки број из тог опсега.

У примеру је изабран број 9999. Када буде модификован овај фајл у постављану на серверски рачунар, треба да се користи порт 80, који је предефинисан за *HTTP* сервер као подразумевани број софтверског порта.



Слика 2.2.2 Део фајла "server.xml" у коме се модификује број порта.

Линије кода од 62 до 68 су коментари у фајлу "server.xml". Број који треба да се промени налази се у линији 69, где уместо "8083" треба да буде "9999", као што је показано на слици 2.2.3.



Слика 2.2.3 Део фајла "server.xml" у коме је модификован број порта.

На слици 2.2.3 се може уочити и предност коришћења едитора текста *NotePad++*, зато што се маркира и боји различитим бојама текст почев од отвореног знака за опис елемента "<" до затвореног знака за опис " />" , а за елемент "Connector ", у фајлу "server.xml", при чему су посебно обојене бројне вредности атрибута.

2.2.1.3 *Enabling Directory Listing* у "conf\web.xml"

На слици 2.2.4 је приказана део фајла "web.xml" у који треба да се уради измене.

```
38    <!-- The default servlet for all web applications, that serves static      -->
39    <!-- resources.  It processes all requests that are not mapped to other    -->
40    <!-- servlets with servlet mappings (defined either here or in your own    -->
41    <!-- web.xml file).  This servlet supports the following initialization    -->
42    <!-- parameters (default values are in square brackets):                   -->
```

C:\Program Files\Apache Software Foundation\Tomcat 8.5\conf\web.xml - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

web.xml

```
103     <servlet>
104         <servlet-name>default</servlet-name>
105         <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
106         <init-param>
107             <param-name>debug</param-name>
108             <param-value>0</param-value>
109         </init-param>
110         <init-param>
111             <param-name>listings</param-name>
112             <param-value>false</param-value>
113         </init-param>
114         <load-on-startup>1</load-on-startup>
115     </servlet>
```

eXtensible Markup Language file length : 174,021 lines : 4,700 Ln : 1 Col : 1 Sel : 0 | 0 Windows (CR LF) UTF-8 INS

...

Слика 2.2.4 Делови фајла "web.xml" у коме се модификује видљивост садржаја фолдера.

*C:\Program Files\Apache Software Foundation\Tomcat 8.5\conf\web.xml - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

web.xml

```
103     <servlet>
104         <servlet-name>default</servlet-name>
105         <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
106         <init-param>
107             <param-name>debug</param-name>
108             <param-value>0</param-value>
109         </init-param>
110         <init-param>
111             <param-name>listings</param-name>
112             <param-value>true</param-value>
113         </init-param>
114         <load-on-startup>1</load-on-startup>
115     </servlet>
```

eXtensible Markup Language file length : 174,020 lines : 4,700 Ln : 112 Col : 30 Sel : 0 | 0 Windows (CR LF) UTF-8 INS

Слика 2.2.5 Делови фајла "web.xml" у коме је одобрена видљивост садржаја фолдера на серверу.

Коришћењем текст едитора *NotePad++* отворити фајл "web.xml", у "conf" фолдеру *Tomcat* инсталационог директоријума. Треба омогућити да се види садржај директоријума на серверу тако да се промени видљивост списка фајлова ("*listings*") са не ("*false*") у да ("*true*") за подразумевани ("*default*") сервлет. На слици 2.2.5 је приказана промена која је у линији 112.

Ова модификација је погодна за тест система, али не и за радну верзију из сигурносних разлога; ако не постоји фајл који треба да види клијент, потребно је да прегледач прикаже садржај директоријума на серверу коме се приступа преко Интернета. То програмеру

значи да треба да направи одговарајући фајл који клијент треба да види на серверу (на пример *index.html*), или да се адреси странице дода назив фајла који треба да види клијент.

Када се комплетан садржај пребацује на сервер, потребно је урадити нову промену, уместо да видљивост списка фајлова буде тачно ("*true*") да се промени да буде нетачно ("*false*") за подразумевани ("*default*") сервлет.

2.2.1.4 *Enabling Automatic Reload* у "conf\context.xml"

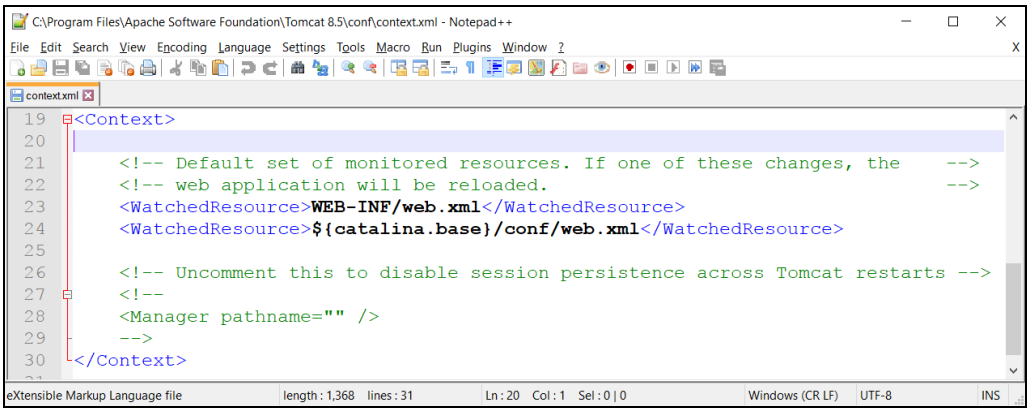
На слици 2.2.6 је приказана део фајла "context.xml" у који треба да се уради измене.

Потребно је да се дода атрибут *reloadable="true"* у <*Context*> елементу да би се аутоматски учитао садржај после промена у коду

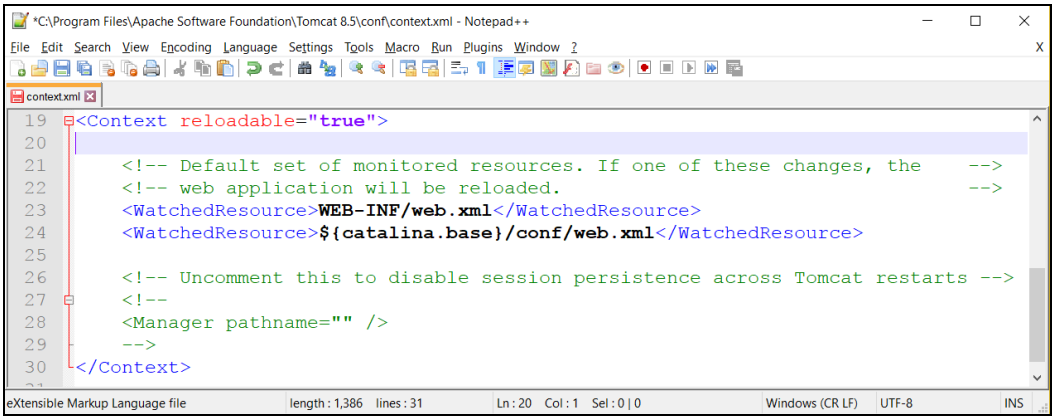
Уместо <Context> треба да буде <Context reloadable="true">.

Ова промена је потребна да би се у тестирању апликације на клијентском рачунару одмах видела промена. Када се пребаци све на серверски рачунар, треба обрисати додату вредност атрибута да би се избегао *overhead* детекције промена

Уместо <Context reloadable="true"> треба да буде <Context>.



Слика 2.2.6 Делови фајла "context.xml" у коме се додаје атрибут елемента <Context>.



Слика 2.2.7 Делови фајла "context.xml" у коме је додат атрибут елементу <Context>.

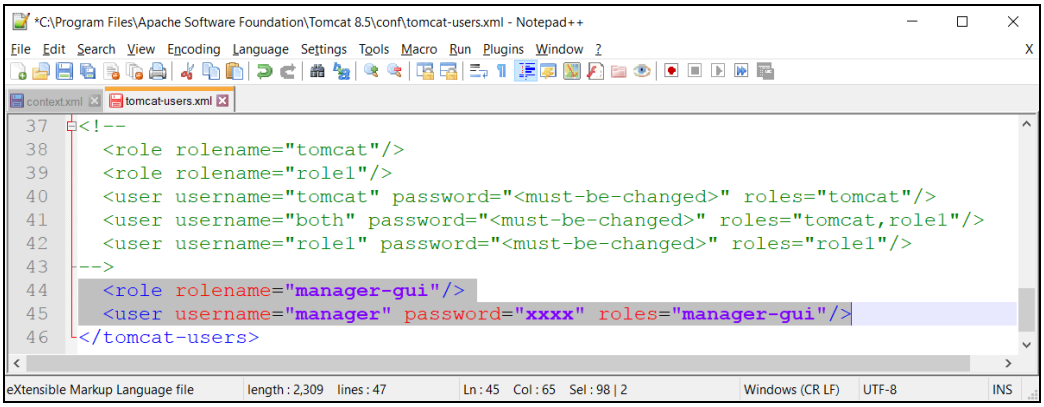
На слици 2.2.7 се види да је урађена измена у линији 19.

Оверхеад (*overhead*) је термин који се користи за било коју комбинацију која превазилази потребе нечега; на пример повећање времена израчунавања, веће коришћење рачунарске меморије, пропусног опсега или било ког ресурса који се захтева да би се извршио неки специфичан посао.

2.2.1.5 Опциона измена у "tomcat-users.xml"

Коришћењем текст едитора (*NotePad++*) отвори се фајл "tomcat-users.xml", у "conf" фолдеру *Tomcat* инсталационог директоријума. Треба унети две линије кода тако да за *Tomcat's manager* буду додате две означене линије унутар <tomcat-users> елемената (између <tomcat-users> и </tomcat-users>), да би се омогућило да *manager GUI app* управља *Tomcat* сервером

```
<tomcat-users>
  <role rolename="manager-gui"/>
  <user username="manager" password="xxxx" roles="manager-gui"/>
</tomcat-users>
```



Слика 2.2.8 Стартовање *Command Prompt* иконице на рачунару.

На слици 2.2.8 су осенчене ове две додате линије као линије 44 и 45 у фајлу.

2.2.1.6 Корак 4, покретање и заустављање рада сервера

За покретање и заустављање рада сервера користићемо у овом курсу команд промпт (*CMD shell, Command Prompt*).

Да би покренули рад сервера потребно је да се пронађе где се сервер налази на рачунару. Обзиром да тестирање рада обављамо на клијентском рачунару, путања до фајла који покреће сервер није у неком подразумеваном фолдеру.

Извршни и скрипт програми који покрећу сервер налазе се у фолдеру "bin" у *Tomcat* инсталационом директоријуму (за *Windows* инсталацију), а то је у нашем примеру

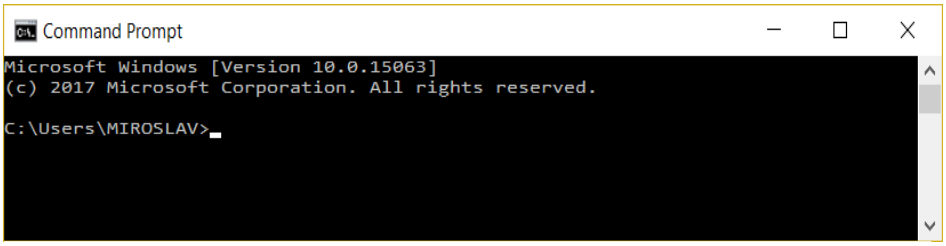
```
"d:\myProject\tomcat\bin"
```

Стартује се команд промпт (*CMD shell, Command Prompt*) и добије приказ као на слици 2.2.9. Отвара се прозор са црном позадином у коме се види шта рачунар враћа као одговор на стартовање и где се уносе команде за даљи рад. Прве две линије показују који је оперативни систем инсталиран на рачунару. Трећа линија приказује у ком фолдеру се тренутно налази курзор за унос команди. То је радни фолдер за који се подразумева да корисник рачунара у њему треба да унесе команду или прими одговор од рачунара, а у наведеном примеру, у овој књизи, то може да буде

```
"C:\Users\korisnik>" или
```

```
"C:\Users\MIROSLAV>"
```

На неком другом рачунару биће другачији текст, тако да уместо "Users" може да буде "korisnik", или уместо "korisnik" да буде корисничко име, на пример "MIROSLAV" ако је више оних који користе рачунар са својим корисничким налогом.



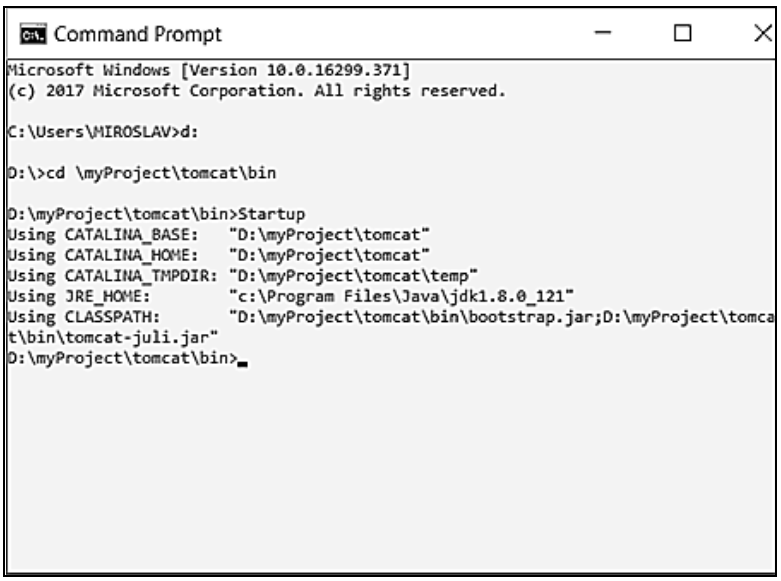

```
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\korisnik>
```

Слика 2.2.9 Два примера прозора након стартовања *CMD*.

Редом треба откуцати у три линије следећи текст

```
d:
cd \myProject\tomcat\bin
Startup
```



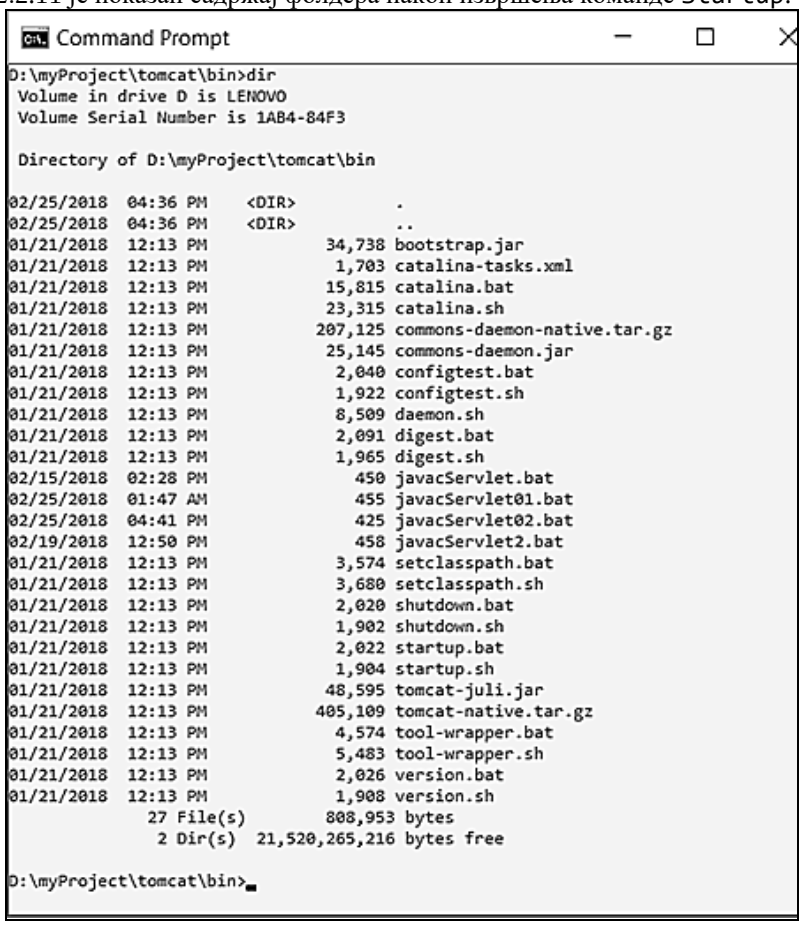
Слика 2.2.10 Покретање сервера из *CMD*.

На слици 2.2.10. је приказан унос ове три линије и шта се дешава. Прва линија кода `d:` пребацује радни директоријум у корени директоријум `D` диска. Друга линија кода (`cd \myProject\tomcat\bin`) пребацује у фолдеру "bin" у *Tomcat* инсталационом директоријуму. Трећа линија кода (`startup`) извршава фајл са екстензијом `.bat` или `.sh` који стартује сервер на рачунару (на пример на слици 2.2.10, `startup.bat`).

Након уноса ових команди добија се извештај шта је рачунар урадио или које су вредности варијабли, као што је податак о томе где се налазе извршни фајлови, где привремене датотеке и извештаји, а где компајлер за програме написане у Јава језику, као и библиотеке које се користе:

```
Microsoft Windows [Version 10.0.16299.371]
(c) 2017 Microsoft Corporation. All rights reserved.
C:\Users\MIROSLAV>d:
D:\>cd \myProject\tomcat\bin
D:\myProject\tomcat\bin>Startup
Using CATALINA_BASE:   "D:\myProject\tomcat"
Using CATALINA_HOME:   "D:\myProject\tomcat"
Using CATALINA_TMPDIR: "D:\myProject\tomcat\temp"
Using JRE_HOME:        "c:\Program Files\Java\jdk1.8.0_121"
Using CLASSPATH:"D:\myProject\tomcat\bin\bootstrap.jar;D:\myProject\tomcat\bin\tomcat-juli.jar"
D:\myProject\tomcat\bin>
```

На слици 2.2.11 је показан садржај фолдера након извршења команде `Startup`.



Слика 2.2.11 Садржај фолдера "bin" у *Tomcat* instalacionom директоријуму.

На слици 2.2.12 је показано да се командом (`shutdown`) прекида рад сервера.

Слика 2.2.12 Део Прекид рада сервера из *CMD*.

Уместо да се сваки пут када се стартује *CMD* куцају команде са тастатуре, уместо да се уносе линија по линија, све команде могу да се напишу у једном фајлу и изврше из тог фајла.

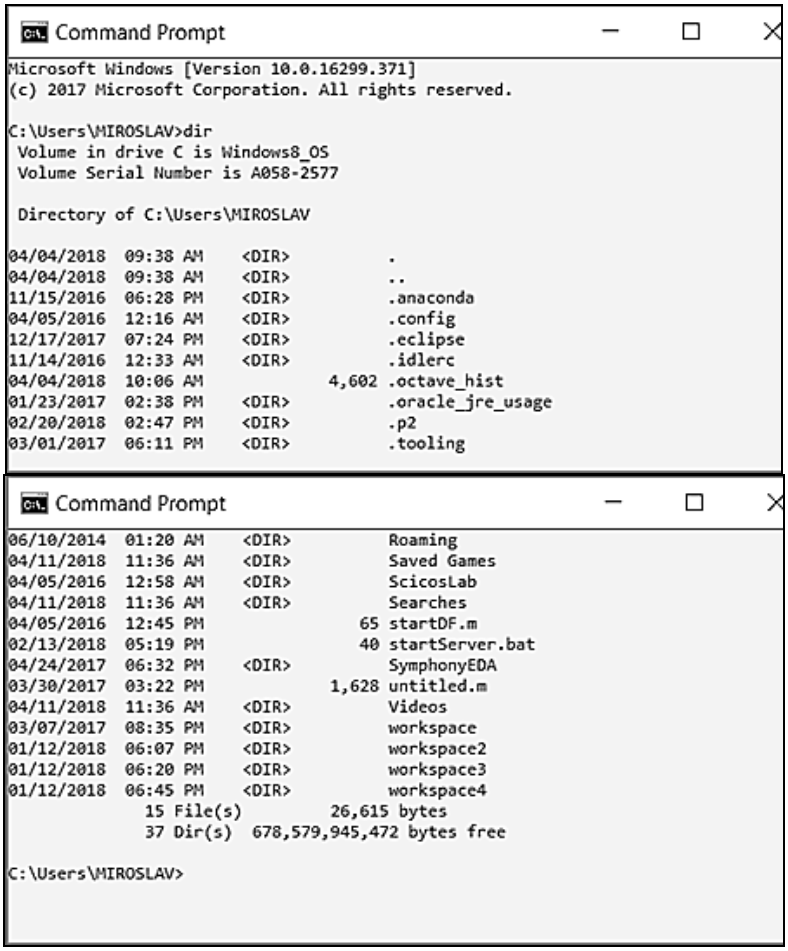
Након што се откуца *exit* у *CMD*, затвара се *CMD* прозор. Сада можемо да урадимо алтернативни начин стартовања сервера. Поново се кликне на црну икону на таскбару и отвори *CMD* прозор.

На слици 2.2.13 је приказан садржај радног директоријума добијен куцањем команде "dir". Приказан је први и последњи део садржаја радног директоријума. Може да се уочи да постоји фајл са именом "startServer.bat", који као објашњење има да је у питању *Windows Batch File*. Садржај фајла је исти као да смо куцали редом линија по линија све ове команде:

```
d:
cd \myProject\tomcat\bin
startup
```

Ако не постоји овај фајл, коришћењем Ноутпед едитора може да се направи фајл са оваквим називом и садржајем.

Сада, када постоји такозвани беч фајл, "startServer.bat", довољно је откуцати део назива лево од тачке, без екстензије, и поново ће бити стартован сервер и добиће се резултати као на сликама 2.2.10, 2.2.11 и 2.2.12. Командом `shutdown` се прекида рад сервера, а када се након тога откуца *exit* у *CMD*, затвара се *CMD* прозор.



Слика 2.2.13 Садржај радног корисничког директоријума

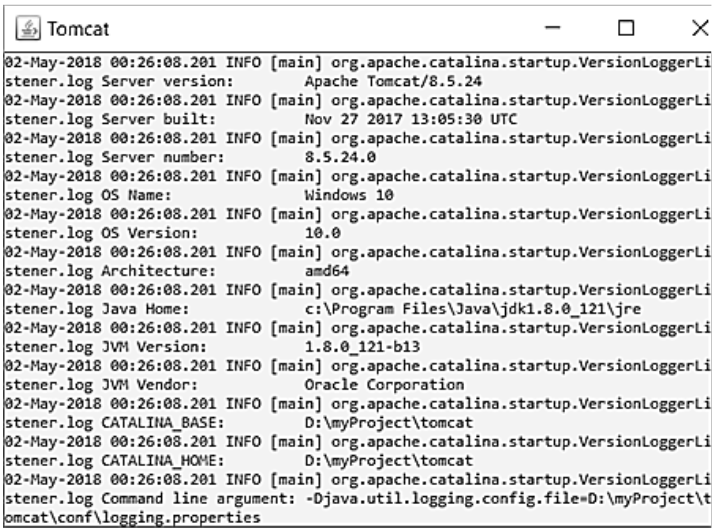
Сваки следећи пут када се покрене *CMD* и откуца "startServer" биће стартован сервер а командом `shutdown` биће прекинут рад сервера; са *exit* у *CMD* затвара се *CMD* прозор. Важно је да имамо ову једноставну процедуру за покретање и заустављање рада сервера, зато што после неких измена у коду, сервер неће увек аутоматски прихватити промене, што програмери тумаче као грешку, па је ово једини начин да будемо сигурни да све ради исправно (слично овоме дешава се код мобилних телефона да се блокира рад неке апликације, и једини начин јесте да се телефон ресетује, или искључи и поново укључи, и тек тада апликација ради исправно).

Сви фајлови који су направљени Ноутпедом остају у фолдерима и могу се користити сваки следећи пут када је то потребно.

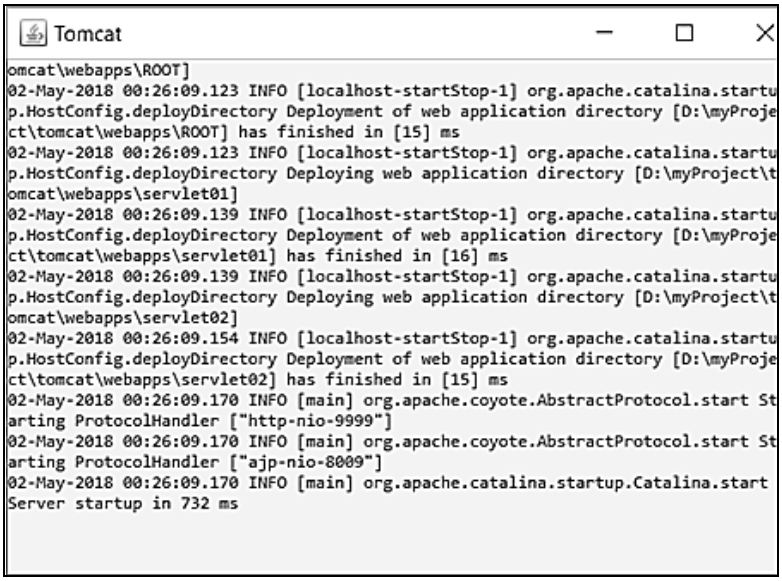
2.2.1.7 Садржај покренутог *Tomcat* сервера

Претпоставимо да смо покренули *CMD* и откуца "startServer", чиме је стартован сервер, а што се манифестује отварањем новог *CMD* прозора; на слици 2.2.14 приказан је први део садржај *CMD* прозора, а на слици 2.2.15 последњи део. Овај прозор се назива серверова конзола, зато што се у њему приказује све што се дешава са сервером (користи се термин *Tomcat console window*). Све будуће грешке биће послате овој конзоли.

У овом новом прозору конзоле треба уочити *Tomcat port* који има број 9999. *System.out.println*() позвани од Јава сервлета биће послати овој конзоли.



Слика 2.2.14 Приказ рада сервера у *CMD*, први део.



Слика 2.2.15 Приказ рада сервера у *CMD*, други део.

Треба уочити да се број порта појављује у овом приказу, као што је приказано на слици 2.2.16.

```
...
...
xxx INFO [main] org.apache.coyote.AbstractProtocol.start
Starting ProtocolHandler ["http-nio-9999"]
xxx INFO [main] org.apache.coyote.AbstractProtocol.start
Starting ProtocolHandler ["ajp-nio-8009"]
xxx INFO [main] org.apache.catalina.startup.Catalina.start
Server startup in 3567 ms
```

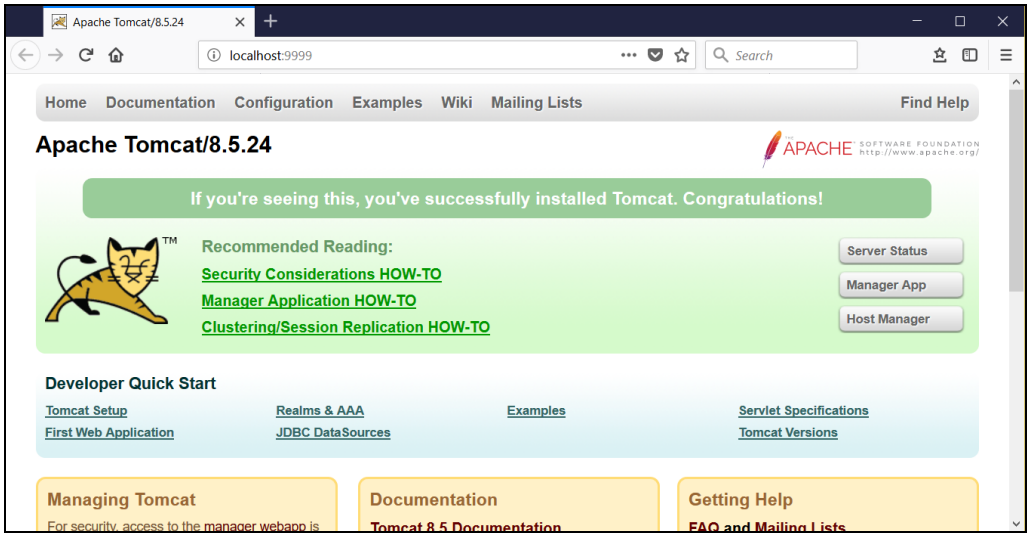
Слика 2.2.16 Приказ дела рада сервера са бројем порта и временом покретања.

Сада кад сервер ради, може се покренути прегледач са клијентове стране и покушати да прикаже шта види на серверу.

2.2.1.8 Покретање прегледача као *HTTP* клијента да приступи серверу

Покретање *Tomcat* прегледача као *HTTP* клијента (на енглеском *Start a browser as HTTP client to access the server*) ради се тако што се покрене прегледач и унесе адреса (*URL*) сервера коме треба приступити, што је у овом случају http://localhost:9999.

Ако је све у реду, треба да се добије *Tomcat server welcome page*, као на слици 2.2.17.



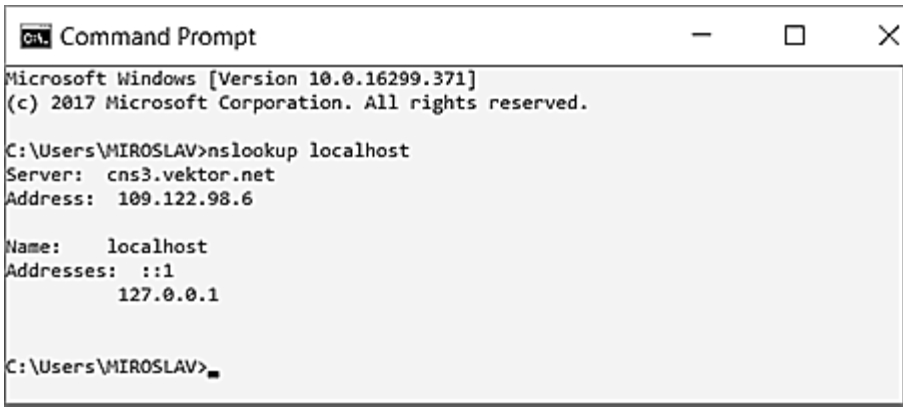
Слика 2.2.17 *Tomcat server welcome page.*

Уместо праве адресе неког рачунара на Интернету, овде се користи *localhost*, а број порта је онај који је подешен као на слици 2.2.3 (:9999).

IP адреса (прецизније *URL* адреса) за "http://localhost:9999" за *Tomcat server's welcome page* је 127.0.0.1, што се може проверити када се искористи команда у *CMD* као што је показано на слици 2.2.18, "nslookup localhost". Име хоста (*hostname*) за "localhost" (са *IP* адресом 127.0.0.1) је намењена за тестирање локалних петљи унутар истог рачунара.

За кориснике са других рачунара преко Интернета, користи се серверова *IP* адреса или DNS име домена или *hostname* у формату, што се формално пише

"http://serverHostnameOrIPAddress:9999".



Слика 2.2.18 *URL адреса за localhost.*

Када се заврши са радом сервера, потребно је да се регуларно искључи.

2.2.1.9 Заустављање сервера

Сервер мора да се заустави на регуларан начин. Никако се не сме насилно прекидати рад сервера, на пример кликом на "CLOSE" дугме, или на знак "✕" у горњем десном углу *CMD* прозора који је отворен након покретања рада сервера као што је био на слици 2.2.15.

Један регуларан начин да се прекине рад сервера јесте да се дође у прозор који је отворен након стартовања сервера а који је на пример приказан на слици 2.34. Овај прозор се назива *Tomcat conzola*. Када се кликне у овај прозор, потребно је да се притисну истовремено типке на тастатури "Ctrl" и "C", односно као јединствена команда "Ctrl-C".

Други начин јесте да се дође у извршни директоријум сервера у *CMD* и стартује фајл за заустављање сервера куцањем са тастатуре `shutdown`, што значи да се извршава `shutdown.bat`. Након тога се откуца *exit* у *CMD* да би се затворио *CMD* прозор.

Да би се извршио фајл `shutdown.bat`, потребно је да је овај фајл у путањи да га препозна оперативни систем, што неће бити ако је у међувремену од покретања сервера промењен директоријум у коме се очекују команде. У том случају је потребно да се понове следеће команде у *CMD* (*Command Prompt*) прозору

```
d:
cd \myProject\tomcat\bin
shutdown
```

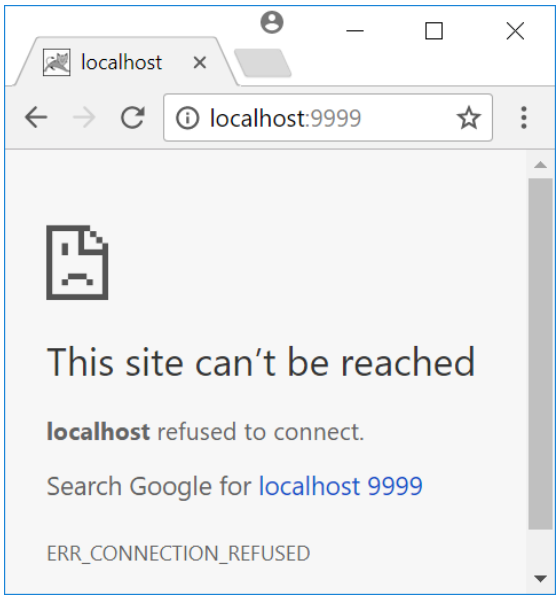
Типична грешка коју програмер може да уради јесте да искључи сервер и након тога покуша да приступи серверу, када се у прегледачу појављује порука грешке. Зато је потребно да се поново стартује сервер.

2.2.1.10 Поновно покретање сервера

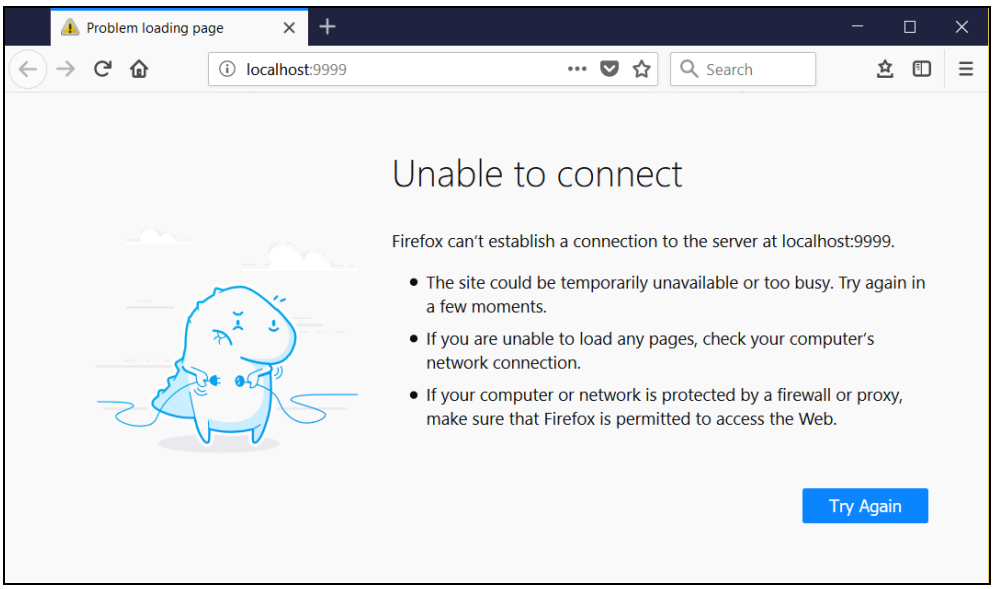
Ако је сервер заустављен са радом тако што је у *CMD* откуцано `shutdown`, или је рачунар био искључен па поново укључен, појавиће се порука

This site can’t be reached
localhost refused to connect.
ERR_CONNECTION_REFUSED

Примери поруке са грешком дати су на сликама 2.2.19 и 2.2.20 за два најпопуларнија прегледача.



Слика 2.2.19 Порука грешке коју јавља прегледач *Google Chrome* након искључења сервера.



Слика 2.2.20 Порука грешке коју јавља прегледач *Mozilla Firefox* након искључења сервера.

Да би се уместо поруке грешке појавио исправан садржај, потребно је поново покренути сервер. Након тога се поново појављује *Tomcat server welcome page*, као на слици 2.2.17.

У случају прегледача *Mozilla Firefox* потребно је кликнути на *Try Again*. У случају прегледача *Google Chrome*, ако се не појави аутоматски *Tomcat server welcome page*, потребно је кликнути на иконицу за рефрешовање стране .

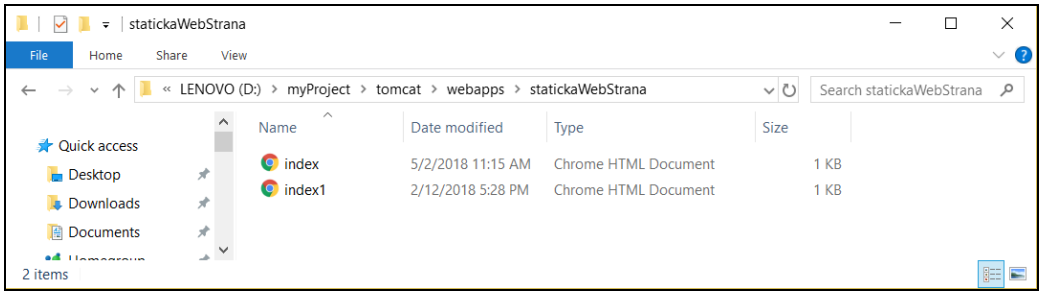
2.2.2 Статичка страна на серверу

За постављање фајлова на серверу морамо да знамо где они треба да се налазе. Подразумевани директоријум за сервер је <TOMCAT_HOME>, који се у нашем примеру налази у d:\myProject\tomcat. Подразумевани директоријум у коме сервер покушава да пронађе фајлове и поддиректоријуме са веб странама које могу да виде клијентски прегледачи је webapps. Директоријум у коме сервер тражи веб апликације на захтев клијентског рачунара који је унео *IP* адреса (*URL* адреса) "http://localhost:9999" у нашој верзији се налази у <TOMCAT_HOME>\webapps, односно на диску на локацији "d:\myProject\tomcat\webapps".

Када се унесе "http://localhost:9999", прегледач подразумева да тражи фајл *index.htm* или *index.php*. Међутим, у оваквој инсталацији, прегледач ће приказати *Tomcat server welcome page*. Зато треба направити нови фолдер за тестирање статичке веб стране.

Направити фолдер у радном директоријуму сервера "statickaWebStrana", као што је показано на слици 2.2.21, тако да је путања до ових фајлова

"d:\myProject\tomcat\webapps\statickaWebStrana"



Слика 2.2.21 Директоријум са статичким веб странама.

Покренемо веб прегледач, и унесемо адресу где очекујемо да добијемо неки садржај,

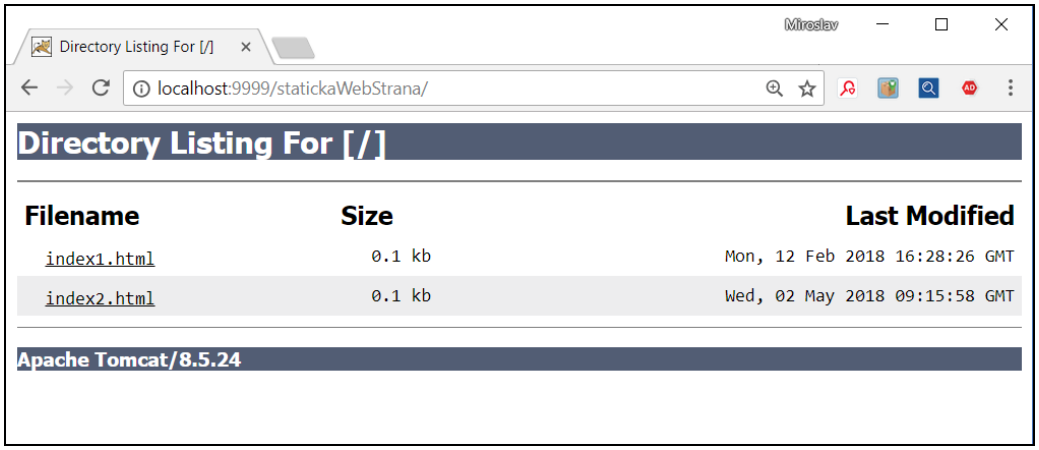
"http://localhost:9999/statickaWebStrana"

Ако није покренут сервер, биће приказана грешка као на сликама 2.2.20 и 2.2.21. Покретањем сервера из *CMD*, у прегледачу би требало да се одмах учита *index.htm*. Аутоматско учитавање је подешено у конфигурацији сервера и није неопходно да буде и на удаљеном серверском рачунару. Приказ који даје прегледач је на слици 2.2.22.

Слика 2.2.22 Прегледач приказује *index.htm* у фолдеру *statickaWebStrana*.

Сада можемо да направимо мали експеримент. Преименујмо фајл *index.htm* у *index2.htm*. У прегледачу се неће ништа променити, он и даље приказује садржај који је добио.

Ако кликнемо на дугме за рефрешовање, прегледач ће приказати садржај фолдера *statickaWebStrana* зато што више нема фајла *index.htm*. На слици 2.2.23 је дат приказ прегледача.



Слика 2.2.23 Прегледач приказује садржај фолдера *statickaWebStrana*.

И ово је било подешено у току конфигурације сервера. Клијент зна да не постоји *index.htm*, али кликом на *index1.htm*, може да види садржај тог фајла (слика 2.2.24). Треба обратити пажњу да текст у прегледачу одговара тексту у телу фајла *index1.htm*, да заглавље приказује то што је у наслову *index1.htm*, а једино *IP* адреса (*URL*) исправно приказује који се фајл приказује у прегледачу.

Слика 2.2.24 Прегледач приказује *index1.htm* у фолдеру *statickaWebStrana*.

У току развоја статичких веб страница може се директно приступити фајловима у фолдеру "statickaWebStrana", као што је показано на слици 2.2.25, тако да је путања до ових фајлова "d:\myProject\tomcat\webapps\statickaWebStrana".

Слика 2.2.25 Прегледач приказује *index2.htm* из *statickaWebStrana* без покретања сервера.

То значи да се за развој не мора користити сервер, већ се ради са фајловима на рачунару, а за тестирање рада сервера је неопходно стартовати рад сервера.

2.2.3 Питања и задаци за вежбу

1. Урадити сва подешавања *Apache Tomcat* сервера тако да се добије поздравна страна сервера.
2. Направити фолдер и у њему фајл са личним подацима студента, укључујући неку слику, а затим прегледачем приказати шта се види на серверу са стране клијента.
3. Променити неке од опција из конфигурације сервера и уочити ефекте приказа прегледача.
4. Проверити какве се поруке добијају, или прикази прегледача, када нису унети тачни подаци за назив фолдера или називе фајлова.

2.3 Комуникација између клијента и сервера

Резиме

Циљ овог поглавља је да студенти овладају израдом једноставног пројекта са *Apache Tomcat* сервером, да направе једноставне веб странице са фајлом на серверу. Други циљ је да студенти разумеју комуникацију између клијента и сервера. Трећи циљ је да се студенти упуте на сајтове са којих могу да преузму додатне примере и објашњења за напредне технике рада са серверима, на пример за рад са сервлетима и *JSP*.

Увод у комуникацију између клијента и сервера

Пре него што се студент посвети сервлетима и другим динамичким апликацијама, пожељно је да је овладао инсталирањем сервера, контролом рада сервера и израдом једноставних (статичких) веб страна, што је приказано у претходним поглављима. Да би у следећем блоку студент могао да прати специфичности динамичких страна, неопходно је да је у потпуности овладао са првим блоком тема.

2.3.1 Израда пројекта са *Apache Tomcat* сервером

Основни задатак после првог блока предавања јесте да се направи неколико статичких веб страна које су међусобно хиперлинковане (са једне стране може да се отвори друга или више њих и да клијент може да обиђе све стране које су постављене на серверском рачунару).

Подразумева се да се и клијентски и серверски рачунар налазе на истом рачунару и да није неопходан приступ интернету током израде пројекта.

2.3.1.1 Прављење једноставне веб странице са фајлом на серверу

Први задатак је да се пронађе фолдер на серверу где клијент покушава да нађе неки садржај, и да посматра шта све приказује прегледач на клијентској страни. Ово је важно да би програмер уочио разлику у приказима прегледача, и ако се касније појаве такви случајеви, да их исправно тумачи и исправи евентуалне грешке.

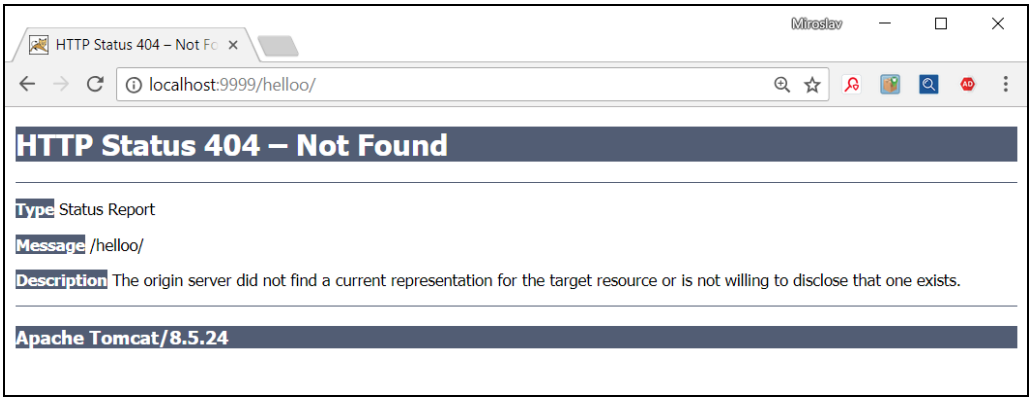
Не сме се заборавити да се прво стартује сервер, па се покрене прегледач и унесе адреса са које треба да прикажемо садржај у прегледачу.

Због могућих грешака, на пример да се заборави да се стартује сервер или унесе непостојећа адреса, добро је да се изврше поступне провере, на пример да се унесе адреса сервера када треба да се добије поздравна порука као што је показано у претходном поглављу за унос "http://localhost:9999".

Као што је показано у претходном поглављу, нека постоји у серверовом директоријуму "webapps" поддиректоријум "statickaWebStrana", тада постоји и директоријум "d:\myProject\tomcat\webapps\statickaWebStrana". У прегледач може да се унесе адреса која садржи назив поддиректоријума "statickaWebStrana", па овај назив може да се дода адреси сервера, "http://localhost:9999/statickaWebStrana/"; добија се садржај фолдера "statickaWebStrana" или приказ на основу *index.htm* који је у њему.

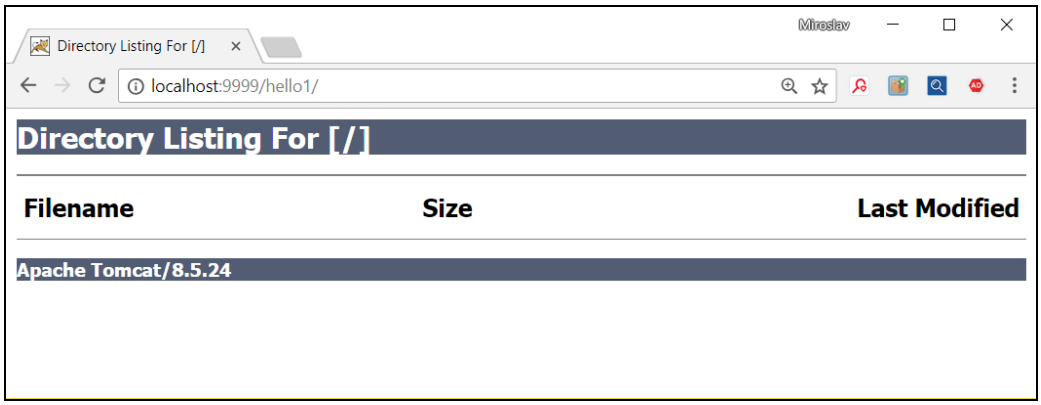
Претпоставимо да клијент зна да користан садржај може да пронађе на серверу тако што одмах после написане адресе сервера ("http://localhost:9999") стоји и додатни текст ("/helloo"), односно да клијент уноси адресу "http://localhost:9999/helloo". У стварној ситуацији уместо "localhost " писаће стварна адреса сервера, а уместо ":9999" ће бити опционо софтверски порт; претпоставимо да ће "/helloo" бити негде на серверу.

Први случај је да је унета непостојећа адреса http://localhost:9999/helloo, што значи да не постоји "d:\myProject\tomcat\webapps\helloo". Прегледач јавља да не постоји то што клијент тражи и враћа поруку као на слици 2.3.1.



Слика 2.3.1 Приказ прегледача када нема одговарајућег директоријума.

Други случај је да је унета постојећа адреса http://localhost:9999/hello1, што значи да постоји "d:\myProject\tomcat\webapps\hello1", али фолдер нема ни један фајл. Прегледач јавља да је пронашао фолдер али не приказује ни један фајл зато што је фолдер празан, као на слици 2.3.2.

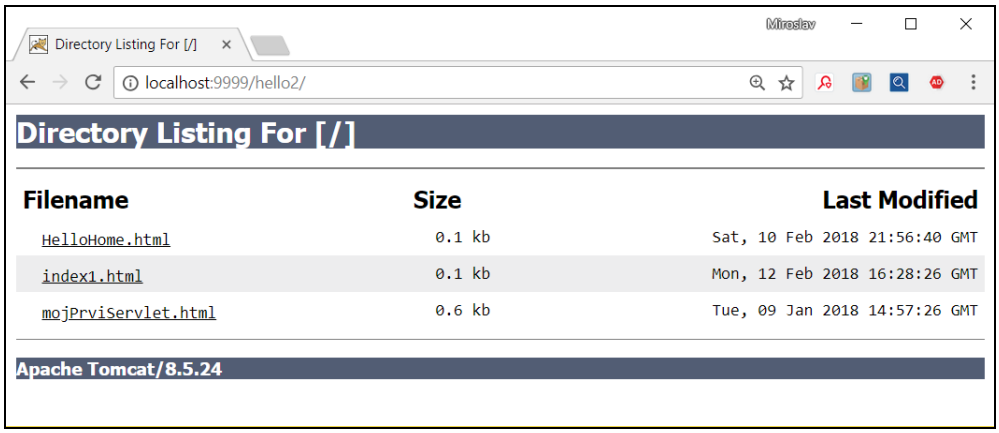


Слика 2.3.2 Приказ прегледача када фолдер постоји али је празан.

Трећи случај који илуструјемо има фајлове у одговарајућем фолдеру и унета је постојећа адреса "http://localhost:9999/hello2", што значи да постоји
"d:\myProject\tomcat\webapps\hello2",

и у фолдеру постоје фајлови и поддиректоријуми али не и *index.htm*. Прегледач јавља да је пронашао фолдер, приказује садржај фолдера, као на слици 2.3.3.

Четврти случај који има фајлове у одговарајућем фолдеру и унета је постојећа адреса "http://localhost:9999/hello3", што значи да постоји "d:\myProject\tomcat\webapps\hello3", и у фолдеру постоји *index.htm* као и више других фајлова и поддиректоријуми. Прегледач јавља да је пронашао фолдер, приказује садржај *index.htm*, као на слици 2.3.4.



Слика 2.3.3 Приказ прегледача када фолдер постоји и има неколико фајлова и фолдера.

Слика 2.3.4 Приказ прегледача када фолдер постоји и има *index.htm.*

У петом случају је унета постојећа адреса "http://localhost:9999/hello/HelloHome.html", што значи да постоји фајл "d:\myProject\tomcat\webapps\hello/HelloHome.html", Прегледач јавља да је пронашао фајл и приказује садржај *HelloHome.html*, као на слици 2.3.5. Уколико не постоји *.html* фајл, прегледач може да преусмери на неку од апликација за проналажења фајлова на Интернету.

Слика 2.3.5 Приказ прегледача када фолдер постоји и има *HelloHome.html.*

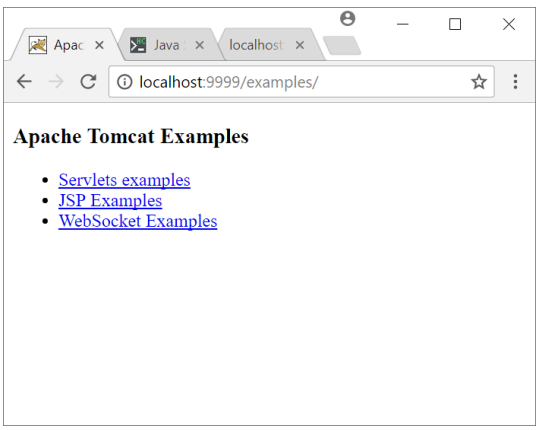
Ради прегледности даје се и садржај фајла *HelloHome.html* који се налази у основном апликационом директоријуму *<TOMCAT_HOME>\webapps\hello;* прегледач може да види ову страну на адреси http://localhost:9999/hello/HelloHome.html:

```
<html>
  <head><title>My Home Page</title></head>
  <body>
    <h1>My Name is so and so. This is my HOME.</h1>
  </body>
</html>
```

2.3.1.2 Додатне информације са сервера

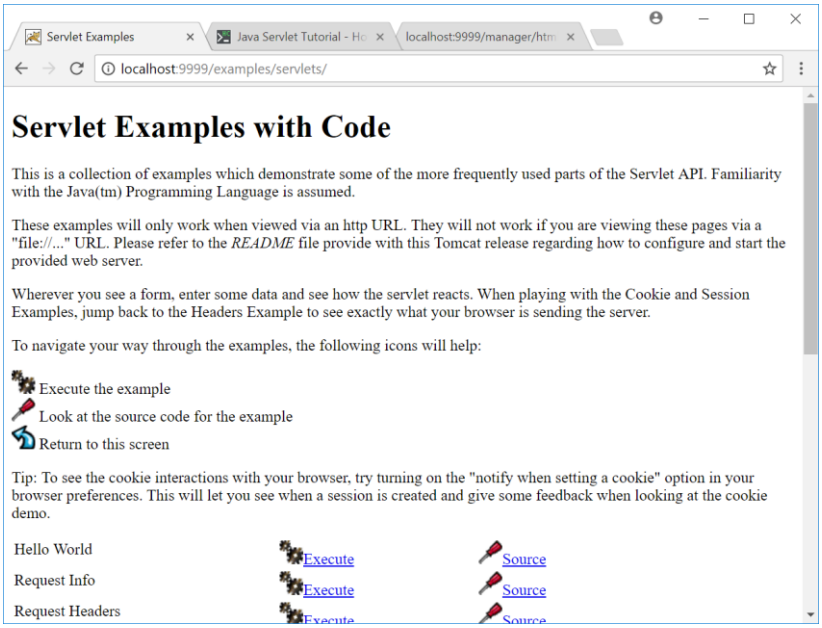
У основном апликационом директоријуму сервера *<TOMCAT_HOME>*, односно у нашем примеру у фолдеру "d:\myProject\tomcat", налазе се још неки фолдери где су додатни примери и објашњења, којима може да се приступи са поздравне стране сервера.

Са *URL* адресом у прегледачу http://localhost:9999/examples могу се видети примери за сервлет и *JSP*, као што је илустровано на слици 2.3.6. Примери се могу покренути на серверу.

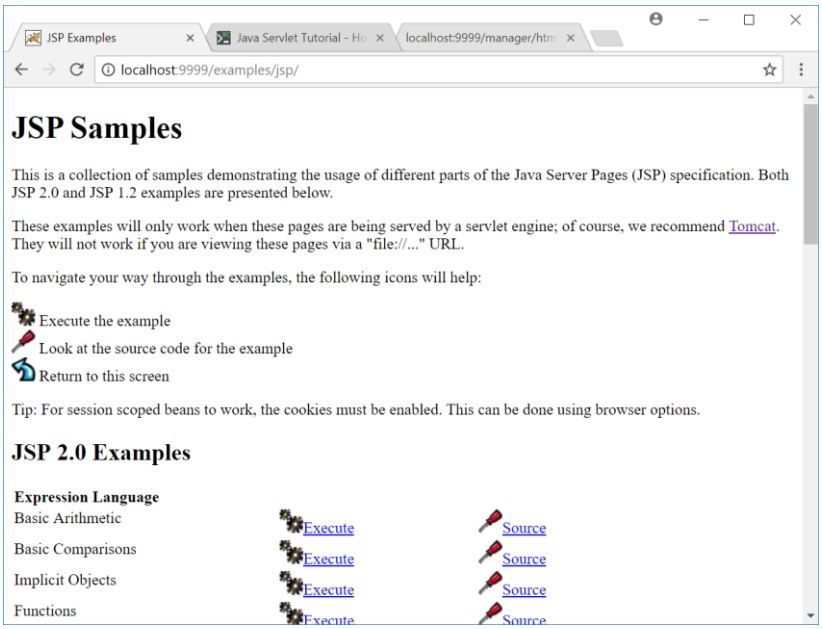


Слика 2.3.6 Сервер примери

Са ове стране се могу покренути сервлет примери, као на слици 2.3.7. Могу се покренути *JSP* примери, слика 2.3.8.

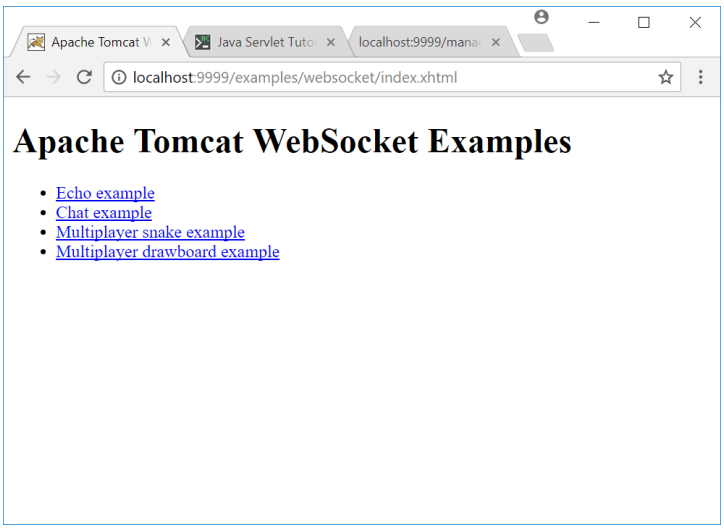


Слика 2.3.7 Сервлет примери

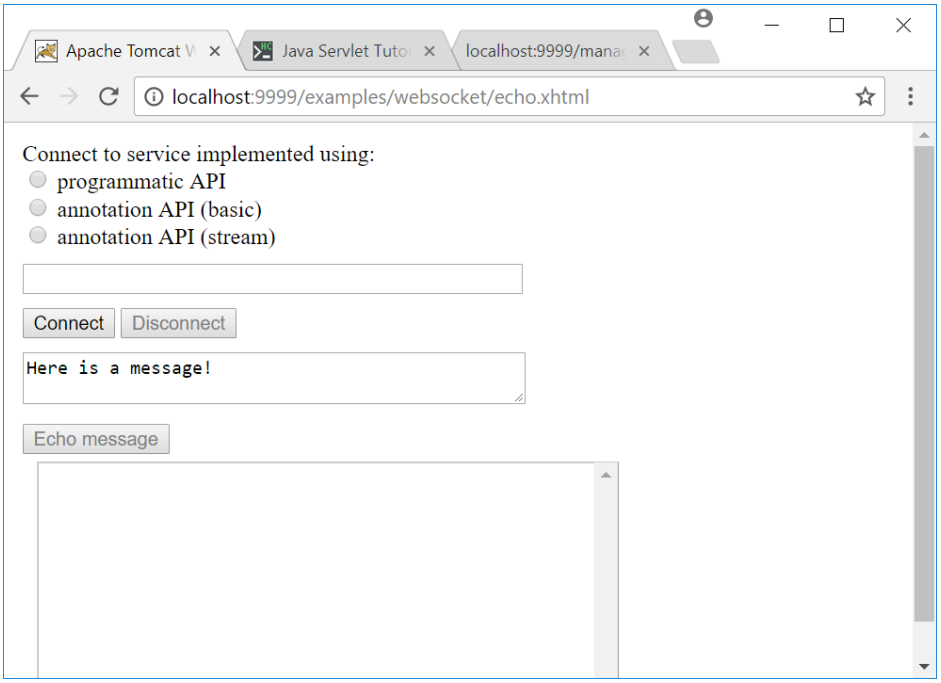


Слика 2.3.8 Примери за *JSP*

На слици 2.3.9 је страница за примере за *WebSocket*. На слици 2.3.10 је страница за примере за *Echo*.

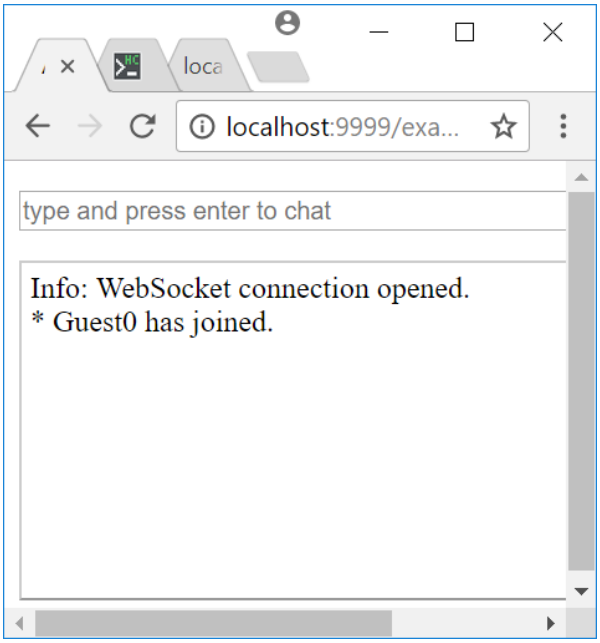


Слика 2.3.9 Примери за *WebSocket*

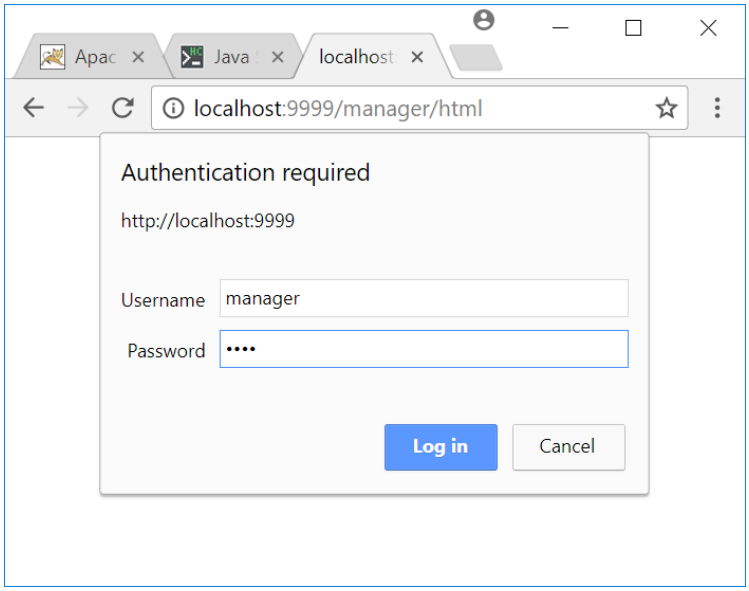


Слика 2.3.10 Примери за *Echo*

На слици 2.3.11 је страница за примере за *Chat*. На слици 2.3.12 је страница за примере за *manager*.



Слика 2.3.11 Примери за *Chat*



Слика 2.3.12 Примери за *http://localhost:9999/manager/html*

Треба се сетити да је код конфигурације сервера http://localhost:9999/manager/html у фајлу "tomcat-users.xml" било написано

```
username="manager"
password="xxxx
```

Потребно је кликнути на дугме Log in in.

2.3.2 Питања и задаци за вежбу

1. Направити фолдер у апликационом директоријуму сервера у који треба направити најмање три фајла који су узајамно повезана и тестирати рад.
2. Тестирати примере из примера који се нуде преко сервера.
3. Написати извештај у форми семинарског рада са илустрацијом урађених примера и описати евентуалне проблеме у развоју и реализацији примера на серверу.

3 *JavaScript*

Резиме

Циљ овог поглавља је да студент кроз примере савлада основне и напредне технике програмирања коришћењем *JavaScript*-а. Студенти ће се упознати са основним и напредним концепти програмског језика који се користи у оквиру *HTML*-а, а за потребе коришћења приказа преко веб прегледача. *JavaScript* је програмски језик је једноставан за учење и има велику примену у пракси.

3.1 Основни појмови

Резиме

Циљ овог поглавља је да студент разуме основе *JavaScript*-а. Кроз једноставне примере, студент треба да се оспособи да савлада рад у овом језику и да разуме карактеристике *JavaScript*-а, како се убацује *JavaScript* код у *HTML* документ, како се користи екстерни *JavaScript* код у *HTML* документу, како се приказују излазни резултати *JavaScript*-а у прегледачу. Студент треба да савлада *JavaScript* наредбе, синтаксу, уношење коментара, декларисање променљивих, коришћење оператора и аритметике, како се користе оператори доделе, који типови података постоје у *JavaScript*-у. Циљ је и да се овлада функцијама, особинама и методама објеката, догађајима, особинама и методама стрингова, особинама и методама за бројеве.

Увод у основе *JavaScript*-а

JavaScript се користи да превазиђе недостатак динамичке обраде унетих података од стране корисника, да разреши проблем честе клијент-сервер комуникације и олакша унос података у неодговарајућем формату (датум рођења, имејл адреса, матични број из личне карте) тако што спречава пренос података од клијента ка серверу за које се на серверу може утврдити да су у неодговарајућем формату. Уместо да сервер проверава тачност, одмах на страни корисника се пријављује грешка и клијент одмах упозорава које податке треба кориговати. На овај начин се избегава непотребан пренос података преко Интернета, кашњење у пријављивању грешака при уносу података због преноса података

од клијента до сервера и одговора сервера које приказује грешку на клијентској страни и указивање на место где је настала грешка. У овом поглављу биће објашњено шта се ради на клијентској страни: како се форматирају подаци, како се обрађују подаци и како се динамички извршава страница у веб прегледачу.

3.1.1 Карактеристике *JavaScript*-а

Основне особине *JavaScript*-а су да је то објектно базирани језик, платформски неутралан језик, вишекориснички језик и програмеру омогућава функционалност на клијентској страни. На пример, на клијентској страни се добије захтев за корекцију одмах када се појави погрешан унос и пре слања серверској страни.

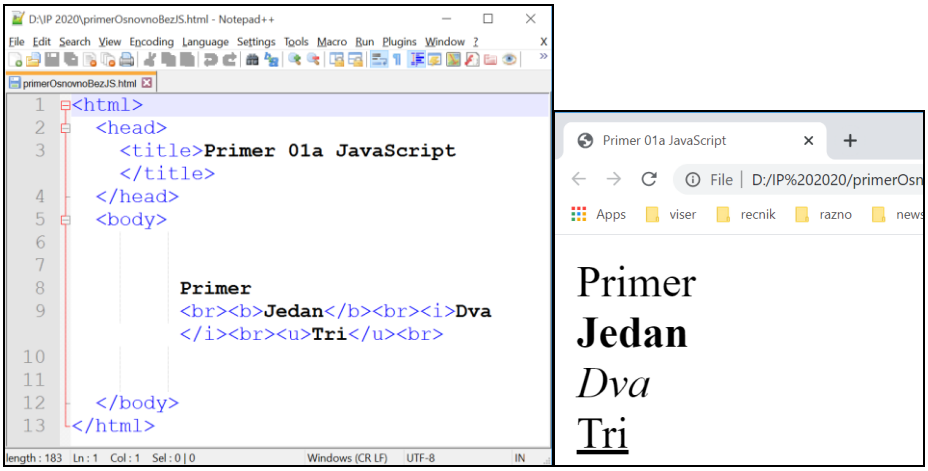
Појам да је *JavaScript* објектно базирани значи да нису реализовани сви концепти објектно оријентисаних језика. На пример, *JavaScript* има лимитиран рад са наслеђивањем и ограниченим важењем и функционалношћу самих објеката. У *JavaScript*-у постоји хијерархија уграђених објеката и они се могу користити са већ дефинисаним методама и особинама. Овиме је добијено је на једноставности самог језика, а помоћу уграђених објеката није изгубљена потребна функционалност. За *JavaScript*-а се каже да је платформски неутралан језик (као *HTML*) што значи да ако је код писан по стандарду, програм би требало да се извршава у оквиру прегледача клијента без обзира која је врста хардвера у рачунару или које софтверско окружење се користи. У време када је настао *JavaScript* било је важно да величина програма буде мала због малих брзина преноса и заузимање протока, што може и данас да буде важно многим апликацијама. Захваљујући томе може да се извршава и на рачунарима са лошијим перформансама.

JavaScript омогућава модуларно програмирање тако што се креирају своји сопствени објекти, дефинишу се опште функције које ће реализовати уобичајене задатке и чувати и извршавати код помоћу посебних докумената. Може да се користи у оквиру тела *HTML* документа, али може да буде и у посебном фајлу са екстензијом .js. Када се користи као посебан .js фајл. Може да се позива и изврши више пута током апликације што значајно смањује обим кода и лакше се одржава софтвер.

Ако се реализује као спољашњи .js документ који ће садржати неку функцију, а која као аргумент прихвата унети текст, на различитим местима употребе је довољно само позвати реализовану функцију. Ако се нешто мења, све промене се извршавају само на једном месту, у екстерном .js фајлу (*JavaScript* документу).

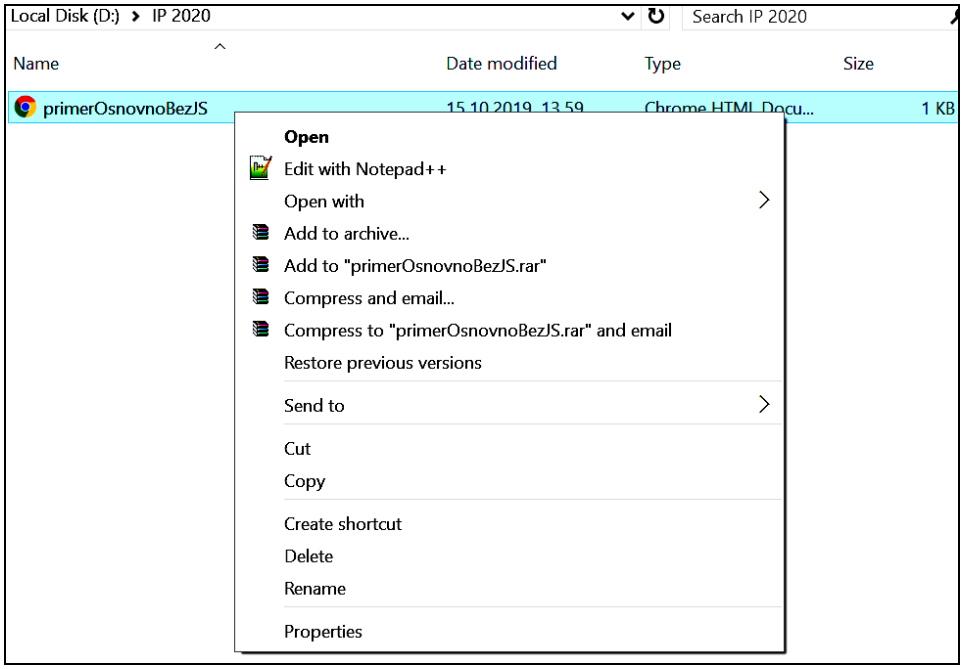
Када се каже да је *JavaScript* интегрисан са *HTML*-ом подразумева се да је у оквиру једне *HTML* стране могуће на произвољан начин комбиновати *JavaScript* и *HTML* код. Из *JavaScript*-а могуће је генерисати сам *HTML* код, у зависности од акције корисника.

На слици 3.1.1 приказан је код *HTML* фајла без *JavaScript*-а као почетни приказ да би се уочиле разлике у односу на *HTML* у који је уграђен *JavaScript*.



Слика 3.1.1 *HTML* без *JavaScript*-а.

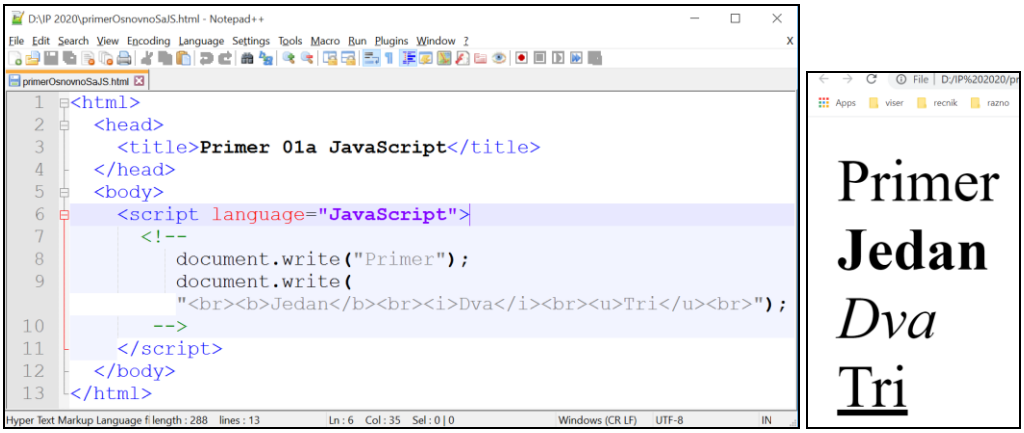
Нека је направљен фајл primerOsnovnoBezJS.html. Овај фајл може да се отвори са Notepad++, тако што се десним кликом миша на фајл добије могућност да се изабере Notepad++ као едитор текста за отварање, и изабере едитор кликом на иконицом . Иако је иконица која је придружена уз фајл *Google Chrome* , за модификовање и генерисање кода се користи едитор текста. Двоструким кликом на фајл, фајл интерпретира и приказује подразумевани прегледач, који је у овом случају *Chrome* као што је приказано на слици 3.1.2 десно.



Слика 3.1.2 Отварање *HTML* фајла са Notepad++.

У примеру са слике 3.1.1 се види да бланко линије 6, 7, 10 и 11 немају утицај на приказ у прегледача, односно прегледач игнорише ове линије. Међутим, линија број 9 се простире у неколико редова, тако да је линија број 10 померена на доле. Најједноставнији начин рада јесте да се отвори посебан фолдер који има једноставну и кратку путању, на пример као у илустрацијама на сликама `D:\IP 2020`. Десним кликом мишем на фајл који има минимум линија, на пример који садржи линије 1 до 5 и 12 и 13 са слике 3.1.1, и већи број бланко линија направи се подразумевани .html фајл; запамти се фајл под другим именом и сачувају промене; затим се унесе код за *JavaScript*; двоструким кликом на фајл стартује се подразумевани прегледач, на пример *Chrome*, и у новом прозору се посматра како је интерпретирао прегледач садржај .html фајла; ако се не добија то што је био циљ, уради се корекција у едитору текста, запамти се под истим именом измењен садржај, а у прегледачу се кликнути на иконицу за рефрешовање стране освежи садржај кликом у *Chrome*-у.

На слици 3.1.3 у левом делу унет је део кода у *JavaScript*-у, и запамћен под другим именом. У десном делу исте слике је приказ прегледача када се фајл отвори са *Chrome*-ом.



Слика 3.1.3 *HTML* са *JavaScript*-ом.

Приказ је идентичан као и у примеру без *JavaScript*-а, приказаног на слици 3.1.1. У овом примеру се уочава да је у телу .html документа промењен садржај тако што се уместо текста написаног у *html* језику, такав текст генерише извршавањем *JavaScript* кода **`document.write()`** где је у загради текст који ће заменити *JavaScript* код текстом између двоструких наводника. Као и у претходном примеру све бланко линије или бланко карактери ће бити игнорирани од прегледача.

Још неколико важних чињеница може да се уочи у овом примеру. Команде *JavaScript* се налазе између знакова за коментар, **`<!--`** и **`-->`** а све је између отвореног тага **`<script language="JavaScript">`** и затвореног тага **`</script>`**. Када прегледач интерпретира шта треба да прикаже у прозору прегледача, он прво долази до отвореног тага за скрипт језик, што за њега значи да мора другачије да интерпретира скрипт фајл од *html* језика. Отворен и затворен знак за коментар значи да не сме да приказује то што је записано као коментар. Када пронађе команду коју разуме *JavaScript*, прво се извршава та команда, која је у овом случају испис текста између двоструких наводника. Како постоје два исписа, цео део између отвореног и затвореног наводника за скрипт текст биће замењени текстом између наводника, па се добија *html* фајл као да је генерисан *html* код са слике 3.1.1.

Наравно, овај једноставан пример је генерисан да би се разумела основна примена *JavaScript*, а шта све може да се уради постаће јасније тек када сеураде други примери.

На слици 3.1.4 упоређена су три примера у којима су у *JavaScript*-у генерисани *html* кодови са свим отвореним и затвореним таговима (**`<b>…</b><i>…</i><u>…</u>`**) и два примера у којима су изостављени затворени тагови **`</b>`** и **`</b>`** и **`</i>`** као илустрација да све функционише на исти начин као да је одмах написан *html* код:

```
document.write("<br><b>Jedan</b><br><i>Dva</i><br><u>Tri</u><br>");
document.write("<br><b>Jedan<br><i>Dva</i><br><u>Tri</u><br>");
document.write("<br><b>Jedan<br><i>Dva<br><u>Tri</u><br>");
```

Уколико не постоји затворени таг за подебљани тип слова, или за укошени тип слова, слова остају подебљана или укошена до краја исписа текста.

```
<body>
  <script language="JavaScript">
    <!--
      document.write("Primer");
      document.write("<br><b>Jedan</b><br><i>Dva</i><br><u>Tri</u><br>");
    -->
  </script>
</body>
```

Са свим таговима

Primer
Jedan
Dva
Tri

……<u>…</u>

```
<body>
  <script language="JavaScript">
    <!--
      document.write("Primer");
      document.write("<br><b>Jedan<br><i>Dva</i><br><u>Tri</u><br>");
    -->
  </script>
</body>
```

Нема затворен таг

Primer
Jedan
Dva
Tri

……<u>…</u>

```
<body>
  <script language="JavaScript">
    <!--
      document.write("Primer");
      document.write("<br><b>Jedan<br><i>Dva<br><u>Tri</u><br>");
    -->
  </script>
</body>
```

Нема затворене тагове и

Primer
Jedan
Dva
Tri

……<u>…</u>

Слика 3.1.4 *HTML* са *JavaScript*-ом је робустан на незатворене тагове.

За разлику од *html* кода који је робустан, *JavaScript* тражи стриктније поштовање правила, као на слици 3.1.5.

```
5   <body>
6     <script language=
      "JavaScript">
7       <!--
8         document.write(
          "Primer");
9         document.write(
          "<br><b>Jedan</b><br>
          <i>Dva</i><br><u>Tri<
          /u><br>");
10      -->
11    </script>
12  </body>
```

Једна линија кода у више редова у едитору


```
5   <body>
6     <script language="JavaScript">
7       <!--
8         document.write("Primer");
9         document.write("<br><b>Jedan</b>
10        <br><i>Dva</i>
11        <br><u>Tri</u><br>");
12      -->
13    </script>
14  </body>
```

Код у више редова у едитору генерише грешку

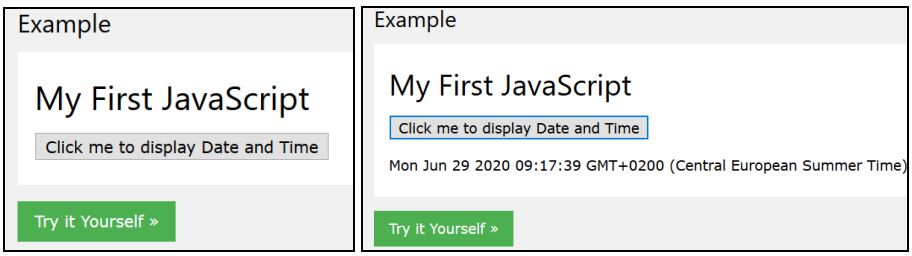
Слика 3.1.5 *HTML* са *JavaScript*-ом у више редова.

У првој врсти је пример где је коректно написан *JavaScript*. Линија 9 кода се простире у приказу едитора текста у више редова, што се јасно види ако се прате бројеви линија у левој колони едитора текста. Сва слова су исте боје између двоструких наводника. У другој врсти је пример где је програмер намерно раздвојио једну линију кода у више линија, где свака линија сада има свој број, од 9 до 11. Одмах се уочава да боја слова у деветој линији није иста као и у 10. и 11. Сада прегледач не може исправно да интерпретира код и зато се ништа не приказује у прозору прегледача. Осим што је неко време активан знак за рефрешовање стране  , после неког времена се појављује знак да је извршавање паузирано.

Сада је јасно да програмер треба да користи едитор текста који има бројеве линија, да би знао да ли је едитор преломио једну линију кода у две или више, или се ради о више врста једне линије кода. Едитор треба да боји одређени тип слова како би се лакше уочило да ли је нека реч команда или само текст; на пример погрешно написана команда у коду биће другачије обојена ако је команда погрешно написана. Када се кликне на линију **`<script>`** или **`</script>`** посебно се обоје ове линије тако да се одмах уочава цео део кода ембедован у *html* код. Још је боље да се користе специјализовани едитори текста који одмах дају предлог текста који треба да се укуца већ после првог слова или неколико првих слова унетих са тастатуре.

У претходним примерима је показано како се у две линије *JavaScript*–а уносе команде које генеришу *html* код. На исти начин је могло да се генерише пет линија *JavaScript* кода са **`document.write("...");`** и тада се не би појављивала грешка у прегледачу.

У наставку текста о *JavaScript*-у, материја ће бити обрађена на основу туторијала објављеног на `https://www.w3schools.com/js`. Како се у оквиру овог курса инсистира на практичној примени, градиво ће бити илустровано примерима тако да се уношењем текста у едитор и извршавањем у прегледачу омогући да одмах након измена у едитору провери резултат. Може да се оде на сајт и директно на њему искористи могућност истраживања и учења по принципу "*Try it Yourself*".



Слика 3.1.6 Пример учења по принципу "*Try it Yourself*".

На слици 3.1.6 лево је илустрован поступак учења. Када се прочитају основне инструкције следи пример `Example` са описом (`My First JavaScript`) и инструкцијама за коришћење (`Try it Yourself`). Текст на дугмету сугерише да треба да се кликне дугме да би били приказани датум и време (`Click me to display Date and Time`). Кликом на дугме

добија се приказ на слици 3.1.6 десно. На овај начин се проверава какав резултат треба да произведе програмски код.

На слици 3.1.7 је илустровано шта се добије када се кликне на `Try it Yourself >>`. У левом делу је код који је у едитору текста, а десно је приказ у прозору прегледача. У едитору може да се промени део кода, и да се кликне на дугме на `Run >>`. У десном делу се види резултат који производи код из едитора.

Слика 3.1.7 Пример учења по принципу "*Try it Yourself*".

На слици 3.1.7 је се могу уочити слова различите боје које другачију приказују команде, атрибуте, текст и вредности променљивих. У овом примеру се кликом на дугме које има идентификацију `id="demo"`, исписује текст који следи после линије у којој је `onclick` а који се налази између двоструких наводника. Шта значи текст у овом примеру биће јасније када се касније у овој књизи објасне сви детаљи.

За израду веб страница, осим *HTML*-а који се користи за дефинисање садржаја, *CSS*-а за дефинисање изгледа веб страница, потребно је да се зна и *JavaScript*, да би се добили резултати који се не могу добити *HTML*-ом и *CSS*-ом. Осим за израду веб страница, *JavaScript* се користи и за друге апликације, како на страни клијента тако и на серверској страни.

Важно је напоменути да су *JavaScript* и *Java* потпуно различити програмски језици иако у свом називу имају исту реч *Java*, како у концептуалном смислу тако и у начину коришћења. *JavaScript* је поставио *Brendan Eich* још 1995., а постао је стандардни језик од 1997. Званични назив стандарда је *ECMA*-262, а званично име програмског језика је *ECMAScript*.

Свако ко је заинтересован да добије сертификат може то да уради онлајн преко неких сајтова на интернету. Студенти који добију оцену већу од 6 у овом курсу уз диплому и додатак дипломе добијају сертификат да су успешно савладали градиво и овладали практичним знањем кроз практичне вежбе.

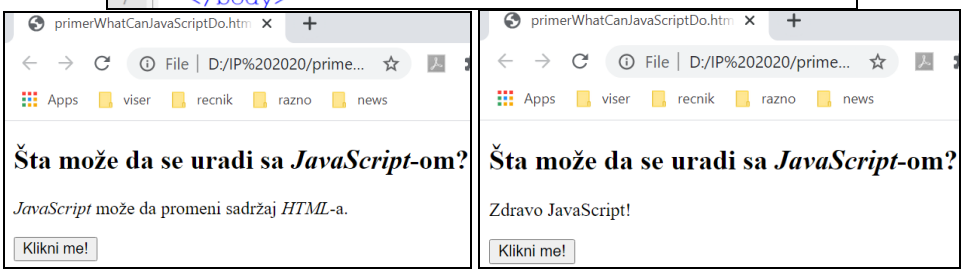
3.1.2 Увод у *JavaScript*

У овом поглављу биће описано кроз примере шта *JavaScript*-а може да уради.

3.1.2.1 Промена *HTML* садржаја

JavaScript може да промени *HTML* садржај тако што у *html* документу генерише текст са *html* таговима. Један од бројних начина јесте да се користи *JavaScript HTML* метода `getElementById()`. У наредном примеру који следи, користи се метода да се "пронађе" један *HTML* елемент (са `id="demo"`) и промени садржај елемента (`innerHTML`) у текст "`Zdravo JavaScript!`": као у примеру са слике 3.1.8.

```
3   <body>
4     <h2>&#352;ta mo&#158;e da se uradi sa <i>
      JavaScript</i>-om?</h2>
5     <p id="demo"><i>JavaScript</i> mo&#158;e da
      promeni sadr&#158;aj <i>HTML</i>-a.</p>
6     <button type="button" onclick=
      'document.getElementById("demo").innerHTML =
      "Zdravo JavaScript!"'>Klikni me!</button>
7   </body>
```



Слика 3.1.8 Промена садржаја *HTML* коришћењем *JavaScript*-а.

У овом примеру можемо да уочимо неколико важних чињеница. Промена садржаја се остварује кликом на дугме Klikni me!. Користи се **`<button type="button" onclick=`**.... Када се кликне дугме долази до промене садржаја на слици 3.1.8 доле-лево у прегледачу на слици доле-десно. Други важан детаљ јесте да се у едитору не користе слова са тастатуре већ *html* ознаке за латинична слова, на пример **`&#158;`** уместо слова ž. Иако се уместо ознаке **`&#158;`** може у едитору користи слово ž, и на клијентском рачунару да се и појави то слово. Међутим, када се код пребаци на сервер и позове са неког другог рачунара *html* фајл, могу се појавити неке друге ознаке уместо слова ž. Због тога се препоручује да се изглед приказа у прегледачу провери у различитим прегледачима, на различитим рачунарима, у различитим верзијама софтвера и апликација.

Да би се избегле непријатности, корисно је користити табелу бројних ознака као што је приказано за нека латинична и ћирилична слова, као и за неке специјалне ознаке:

Ž	Ž		**Ж**	Ж		** **	нераскидиви размак
ž	ž		**ж**	ж		**<**	<, мање од, **<**
Ć	Ć		**љ**	љ		**>**	>, веће од, **>**
ć	ć		**њ**	њ		**&**	&, *ampersand*, **&**
Č	Č		**т**	т		**"**	", знак двоструког навода, **"**
č	č		**з**	з		**'**	', *apostrof*, једноструки наводник
Đ	Đ		**у**	у		**%**	%, проценат
đ	đ		**и**	и		**#**	#, ознака за број
Š	Š		**о**	о		**!**	!, узвичник
š	š		**п**	п		** **	бланко знак, размак, ** **

JavaScript прихвата и једноструке и двоструке наводнике; на пример исти резултат се добија са било којом од две наредне линије кода:

```
document.getElementById("demo").innerHTML = "Zdravo JavaScript";
document.getElementById('demo').innerHTML = 'Zdravo JavaScript';.
```

3.1.2.2 Промена атрибута *HTML* елемента

JavaScript може да промени вредност атрибута у *HTML* документу, на пример атрибут тага за слику где се мења атрибут извора приказивања слике (назив слике у фолдер), као што је показано на слици 3.1.9. Иницијално, када прегледач учита фајл он приказује слику `sijalica0.gif` која има угашену (провидну) сијалицу.

Ако се кликне на Upali sijalicu мења се атрибут `<img ... >` тако да се учитава слика `sijalica1.gif` која има жуто офарбану сијалицу. Визуелно се стиче утисак као да смо кликом на дугме укључили сијалицу. Колико год пута да кликнемо исто дугме, ништа се неће променити зато што се приказује жута сијалица.

Ако се кликне на Ugasi sijalicu мења се атрибут `<img ... >` тако да се учитава слика `sijalica0.gif` која има провидну сијалицу, односно као да је сијалица угашена.

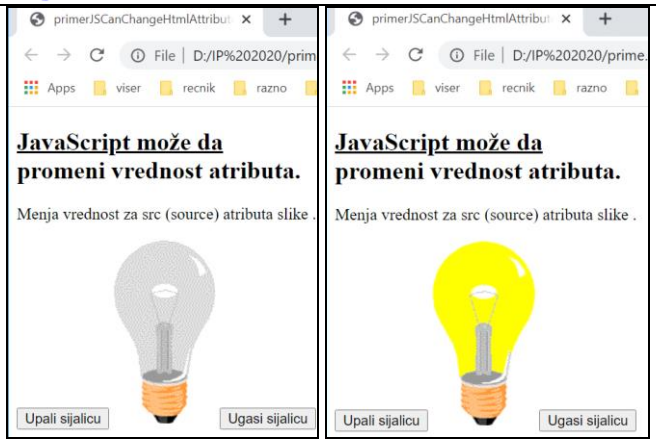
Суштина овог начина промене јесте да се у линији 10 *HTML* кода где је опис да треба приказати слику, употребљен и идентификатор `id="myImage"` који се негде налази у телу документа. Идентификатор се појављује на два места где се налазе два дугмета. У зависности од тога које дугме је притиснуто, мења се атрибут за слику који мења име слике као атрибута. Ако се кликне два пута исто дугме, ништа се неће променити зато што се поставља исти назив слике. Ако се кликне друго дугме, мења се назив слике.

```
<body>
  <h2><u>JavaScript mo&#158;e da</u>
  <br>promeni vrednost atributa.</h2>
  <p>Menja vrednost za src (source) atributa slike .</p>

  <button onclick=
  "document.getElementById('myImage').src='sijalica1.gif'">
  Upali sijalicu</button>

  <img id="myImage" src="sijalica0.gif" style="width:100px">

  <button onclick=
  "document.getElementById('myImage').src='sijalica0.gif'">
  Ugasi sijalicu</button>
</body>
```



Слика 3.1.9 Промена атрибута *HTML* за назив слике.

Важно је да се уочи да у последњем примеру није коришћен отворени таг **<script language="JavaScript">** и затворени таг **</script>**. Све линије се приказују у прегледачу , па нису потребни ни тагови за коментар, **<!--** и **-->**. Два дугмета се увек приказују, а резултат извршавања се мења у зависности од тога да ли је и које је дугме кликнуто. Линија са сликом, **<img ... >**, увек приказује слику, а коју, то зависи од тога да ли је кликнуто неко од два дугмета, или није кликнуто ни једно дугме.

Једна од варијанти промене атрибута јесте да се промени стил као што би се урадило са *CSS*. Пример је дат на слици 3.1.10 где је идентификатор постављен као атрибут у тагу за параграф. Кликом на дугме се проналази под којим идентификатором треба да се уради промена, и која конкретно промена. У примеру се мења величина фонта када се кликне на дугме. Као и у примеру са слике 3.1.9 није потребно користити отворени и затворени таг за **<script>**, као ни линије за коментар.

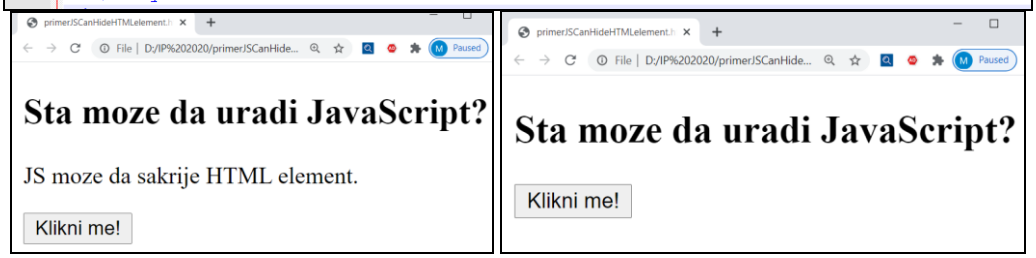
У свим наредним примерима нећемо користити латинична слова српског језика да би се лакше идентификовао садржај *html* фајла у едитору у односу на приказ у прегледачу.

```
<body>
  <h2>Sta moze da uradi JavaScript?</h2>
  <p id="demo">JS moze da promeni stil HTML elementa.</p>
  <button type="button" onclick="document.getElementById('demo').style.fontSize='35px'">
  Klikni me!</button>
</body>
```

Слика 3.1.10 Промена атрибута *HTML* за величину фонта у параграфу.

Пример промене садржаја у прегледачу приказан је на слици 3.1.11 где се кликом на дугме сакрива један *html* елемент. Када се у прегледачу прикаже садржај фајла са слике, приказују се наслов, један параграф и дугме. Кликом на дугме, проналази се идентификатор наведен у опису дугмета, затим линија *html* кода која садржи идентификатор, у овом случају то је атрибут за параграф, коме се мења вредност да не буде приказан (`style.display='none'`). Након клика се поставља приказ у прегледачу где цео параграф није видљив. Као и у претходним примерима, није потребно користити отворени и затворени таг за **`<script>`**, као ни линије за коментар.

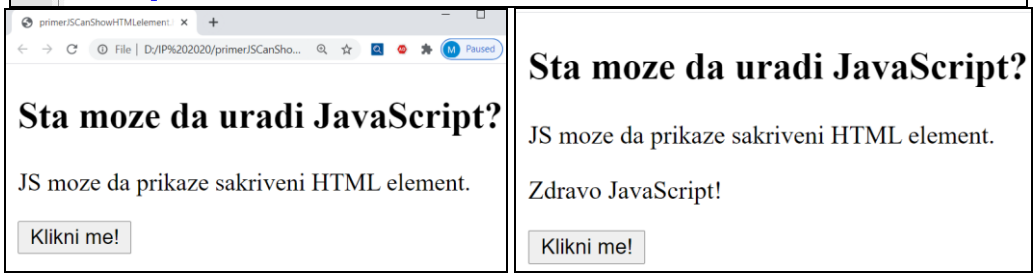
```
<body>
  <h2>Sta moze da uradi JavaScript?</h2>
  <p id="demo">JS moze da sakrije HTML element.</p>
  <button type="button" onclick=
  "document.getElementById('demo').style.display='none'">Klikni me!</button>
</body>
```



Слика 3.1.11 Промена атрибута *HTML* за сакривање параграфа.

На исти начин као што је сакривен цео параграф у примеру 3.1.11, параграф који је био сакривен (`display:none`) може променити вредност атрибута да буде видљив са `style.display='block'`. Након клика на дугме, мења се вредност атрибута за параграф и приказује се додатни параграф JS moze da prikaze sakriveni HTML element. Пример је илустрован на слици 3.1.12.

```
3   <body>
4     <h2>Sta moze da uradi JavaScript?</h2>
5     <p>JS moze da prikaze sakriveni HTML element.</p>
6     <p id="demo" style="display:none">Zdravo JavaScript!</p>
7     <button type="button" onclick=
      "document.getElementById('demo').style.display='block'">Klikni me!</button>
8   </body>
```



Слика 3.1.12 Промена атрибута *HTML* за приказ сакривеног параграфа.

Сада када смо сазнали неке од могућности *JavaScript*-а, можемо да се подсетимо на уобичајена питања: Одакле се преузима *JavaScript*? Где се преузима и смешта *JavaScript*? Како се инсталира *JavaScript*? Да ли је бесплатан *JavaScript* и под којим условима (лиценцом) се може користити *JavaScript*?

JavaScript је већ уграђен у све прегледаче као стандардни део прегледача. Не треба да се преузима или инсталира на рачунарски хардвер, без обзира да ли је то стандарни рачунар, таблет, паметни телефон или рачунар величине кредитне картице. Бесплатан је и није потребна регистрација или преузимање са Интернета.

3.1.3 Убацивање *JavaScript* кода у *HTML* документ

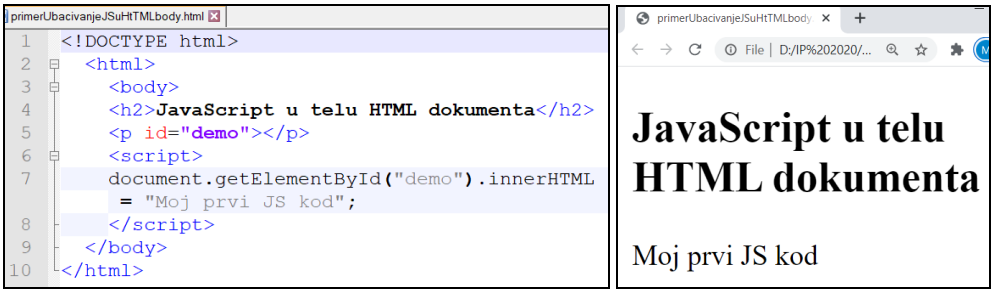
У претходним примерима је показано да се *JavaScript* може инсертовати у *html* документ иако се нигде не наводи да је то скрипт код.

У овом поглављу ће бити објашњено како се све ембедује *JavaScript* код у *html* документ. Најједноставнији начин убацивања *JavaScript* кода је коришћењем скрипт тага. На пример у *HTML*, *JavaScript* код се инсертује између отвореног `<script>` и затвореног `</script>` тага:

```
<script>
  document.getElementById("demo").innerHTML = "Moj prvi JavaScript kod";
</script>
```

Пример убацивања *JavaScript* кода у *HTML* документ, између `<script>` и `</script>` тага, приказан је на слици 3.1.13. Важно је уочити да се у едитору, слика лево, линија 7 кода простире у два реда у *html* документу. У прегледачу се наслов нивоа 2 простире у 2 реда зато што је сужена десна ивица прегледача, иначе би била написана у једном реду. Линија едитора број 5 садржи опис параграфа који треба да буде написан одмах испод наслова који је у линији 4 *html* документа. Прегледач у линији 5 проналази идентификатор `id="demo"`, а да би знао шта треба да уради проналази исти идентификатор у скрипт запису. Скрипт код извршава команду `document.getElementById("demo").innerHTML =`

"...";` То што је између двоструких наводника "`Moj prvi JS kod`" је текст који ће бити убачен између тагова за параграф који су у линији 5 у едитору.



Слика 3.1.13 Убацивање *JavaScript* кода у HTML документ.

У ранијим верзијама *JavaScript*-а користио се и атрибут који је описивао да је скрипт код који следи *JavaScript*: `<script type="text/javascript">`. У садашњим верзијама није потребно да се додаје овај атрибут зато што је *JavaScript* подразумевани скрипт језик у *HTML*-у.

JavaScript функције (*functions*) и догађаји (*events*) су још један начин убацивања *JavaScript* кода. *JavaScript* функције су блок *JavaScript* кодови који могу да се изврше када позову. На пример, функција може да се позове када се деси неки догађај, као што је када се кликне на дугме, а што је показано у примерима пре овог поглавља.

У *html* документу може да буде произвољан број *JavaScript* скриптова. *JavaScript* скриптови могу да буду у телу *html* документа (`<body>`), или у `<head>` секцији, или у оба. На пример, нека је у едитору следећи код:

```
1. <!DOCTYPE html>
2. <html>
3.  <head>
4.    <script>
5.    function myFunction() {
6.    document.getElementById("demo").innerHTML = "Promena Paragrafa.";
7.    }
8.    </script>
9.  </head>
10. <body>
11.   <h1>Veb strana sa JS funkcijom</h1>
12.   <p id="demo">Paragraf</p>
13.   <button type="button" onclick="myFunction()">Klikni</button>
14. </body>
15.</html>
```

То што се налази између тагова **<head>** и **</head>** нема директан утицаја на приказ у прегледачу. Прегледач приказује то што је између тагова **<body>** и **</body>**. Приказује

текст за наслов величином **<h1>**. Затим исписује текст **Paragraf** у параграфу који се види на слици 3.1.14. Атрибут за параграф **<p id="demo">** се игнорише. Следи параграф за дугме. Када се кликне дугме, активира се функција **myFunction()**, Прегледач тражи дефиницију за функцију и проналази је између тагова за **<head>**. Функција прозива идентификатор **"demo"**, у телу између тагова **<body>** проналази где је параграф са овим идентификатором, и текст параграфа у коме стоји **Paragraf** мења са новим текстом **Promena Paragrafa.** који је у дефиницији функције.

Слика 3.1.14 Промена параграфа са функцијом у **<head>**-у и дугметом у телу.

Претходни пример илуструје случај када је функција између тагова за **<head>**, а дугме и садржај између тагова **<body>**. Када се кликне на дугме, текст који је у параграфу се мења текстом који је између двоструких наводника у функцији.

JavaScript скриптови могу да буду само у телу *html* документа (<body>), као што је илустровано у следећем примеру на слици 3.1.15.

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Veb strana sa JS funkcijom u &lt;body&gt;</h1>
    <p id="demo">Paragraf</p>
    <button type="button" onclick="myFunction()">Klikni</button>
    <script>
     function myFunction() {
     document.getElementById("demo").innerHTML = "Promena paragrafa.";
     }
    </script>
  </body>
</html>
```

Слика 3.15 Промена параграфа са функцијом и дугметом у секцији **<body>**.

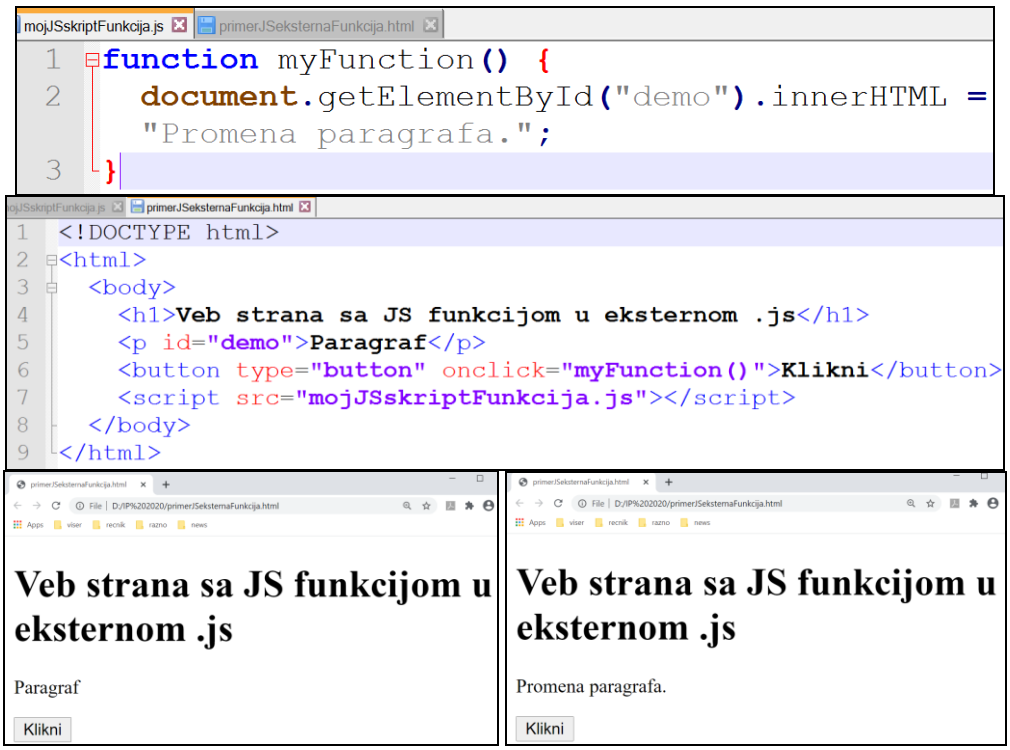
Добија се исти резултат као у случају са слике 3.1.14. Прегледач приказује то што је између тагова **<body>** и **</body>**. Приказује текст за наслов величином **<h1>**. Затим исписује текст **Paragraf** у параграфу. Атрибут за параграф **<p id="demo">** се игнорише.

Следи параграф за дугме. Када се кликне дугме, активира се функција **`myFunction()`**, Прегледач тражи дефиницију за функцију и проналази је између тагова за **`<script>`**. Функција прозива идентификатор **`"demo"`**, у телу између тагова **`<body>`** проналази где је параграф са овим идентификатором, и текст параграфа у коме стоји **`Paragraf`** мења са новим текстом **`Promena paragrafa.`** који је у дефиницији функције. Када се поново кликне дугме, у прегледачу се ништа не мења зато што се са сваким кликом поставља исти текст **`Promena paragrafa.`**.

Постављање скрипти на крај сегмента **`<body>`** побољшава брзину приказа, зато што интерпретација скрипти успорава приказ у прегледачу.

3.1.4 Екстерни *JavaScript* код у *HTML* документ

Део скрипт кода који је био смештен између тагова за **`<script>`**, као у претходним примерима, може да буде у посебном фајлу који има екстензију .js, као што је показано на слици 3.1.16.



Слика 3.1.16 Промена параграфа са функцијом у екстерном .js фајлу.

Текст дефиниције функције из примера са слике 3.1.15 смештен је у посебан фајл са екстензијом .js, `mojJSskriptFunkcija.js`. У *html* фајлу се позива ова функција као атрибут скрипт дела `<script src="mojJSskriptFunkcija.js"></script>`, где се користи име `.js` фајла као извор одакле се учитава фајл.

Овакав начин рада је посебно погодан када се иста дефиниција функције користи у више *html* фајлова или више пута у истом фајлу.

Референца са позивом екстерног `.js` фајла може да буде између тагова за **`<head>`** или између тагова за **`<body>`**. Екстерни скрипт фајл се понаша као да је унесен тачно на том месту у секцији за **`<head>`** или за **`<body>`**.

Екстерни скрипт фајлови не смеју да садрже `<script>` тагове.

Овакав начин раздвајања *html* и екстерних `.js` фајлова олакшава читљивост и одржавање програмског кода. Кеширани *JavaScript* фајлови могу да убрзају учитавање веб стране.

Може се додати неколико екстерних скрипт фајлова који имају различите називе, на пример:

```
<script src="mojaJSskriptFunkcija1.js"></script>
<script src="mojaJSskriptFunkcija2.js"></script>.
```

Екстерни скрипт фајлови се могу референцирати преко *URL* адресе, или коришћењем путање релативно у односу на веб страну; следи пример са пуном адресом,

```
<script src="https://www.w3schools.com/js/myScript1.js"></script>
```

или, релативно у поддиректоријуму `/js` у односу на фолдер у коме се налази *html* фајл који приказује прегледач,

```
<script src="/js/mojaJSskriptFunkcija1.js"></script>
```

Ако се скрипт фајл налази у истом фолдеру у којем је и *html* фајл који приказује прегледач, тада није потребно додавати путању, као што је већ илустровано на слици 3.1.16.

У следећој табели је дато неколико примера путања за извор одакле се преузима референца на слику у `.jpg` фајлу на рачунару.

Путања (path)	**Опис**
`<img src="slika.jpg">`	`slika.jpg` фајл са екстензијом `.jpg` налази се у истом фолдеру као и *html* фајл у текућем фолдеру
`<img src="images/slika.jpg">`	`slika.jpg` фајл се налази у поддиректоријуму `images` у односу на *html* фајл у текућем фолдеру
`<img src="/images/slika.jpg">`	`slika.jpg` фајл се налази у поддиректоријуму `images` у односу на `root` фолдер приказане веб стране
`<img src="../slika.jpg">`	`slika.jpg` фајл се налази један ниво изнад текућег фолдера

3.1.5 Излазни резултати *JavaScript* у прегледачу

JavaScript може да прикаже резултате на више начина:

- Уписивањем у *HTML* елемент коришћењем `innerHTML`.

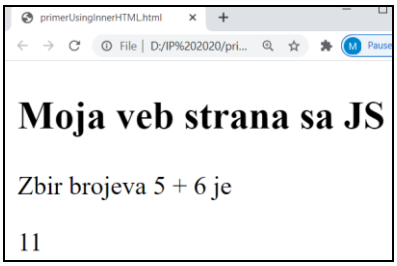
- Уписивањем у *HTML* излаз коришћењем `document.write()`.
- Уписивањем у прозор упозорења коришћењем `window.alert()`.
- Уписивањем у конзолу прегледача коришћењем `console.log()`.

3.1.5.1 Коришћење *innerHTML*

Да би приступио *HTML* елементу, *JavaScript* може да користи методу `document.getElementById(id)`. Атрибут `id` дефинише *HTML* елемент. Особина `innerHTML` дефинише *HTML* садржај:

У примеру приказаном на слици 3.1.17 лево приказан је *html* документ који се види у прегледачу као на истој слици десно. За разлику од претходних примера, у овом се е користи дугме да би се извршио скрипт код.

```
1  <!DOCTYPE html>
2  <html>
3    <body>
4      <h2>Moja veb strana sa JS</h2>
5      <p>Zbir brojeva 5 + 6 je</p>
6      <p id="demo"></p>
7      <script>
8  document.getElementById("demo").innerHTML = 5+6;
9      </script>
10   </body>
11 </html>
```



Слика 3.1.17 Уписивањем у *HTML* елемент коришћењем `innerHTML`.

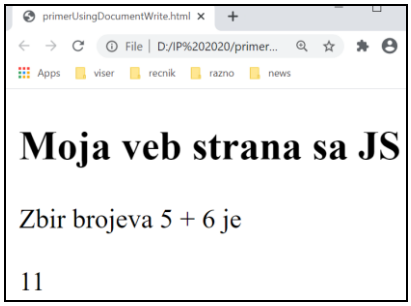
Прегледач редом извршава линије кода у *html* документу; најпре се приказује наслов другог нивоа, па параграф са текстом који најављује шта ће бити приказано, а у следећем параграфу се приказује резултат извршавања према скрипт коду. Када прегледач дође до линије 6, проналази идентификатор; у скрипт коду проналази шта треба да се уради на месту где је идентификатор, а то је уписивање резултата збира два броја у параграфу у коме је идентификатор између тагова за параграф. Резултат да је израчуната вредност 11 је видљив у прегледачу на слици десно.

Уобичајен начин приказа података у прегледачу и промене неке од особина и садржаја *html* елемента јесте коришћењем `innerHTML`.

3.1.5.2 Коришћење *document.write()*

За потребе тестирања изгледа веб стране, корисно је користити `document.write()`. Пример је приказан на слици 3.1.18: лево је *html* документ, десно је приказ у прегледачу.

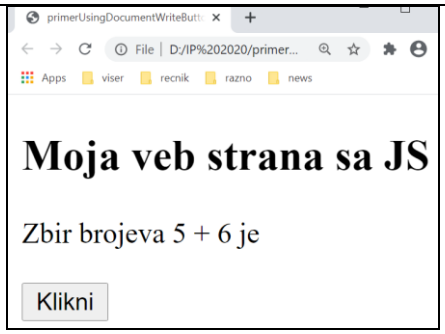
```
<!DOCTYPE html>
<html>
  <body>
    <h2>Moja veb strana sa JS</h2>
    <p>Zbir brojeva 5 + 6 je</p>
    <h3 id="demo"></h3>
    <script>
      document.write(5+6);
    </script>
  </body>
</html>
```

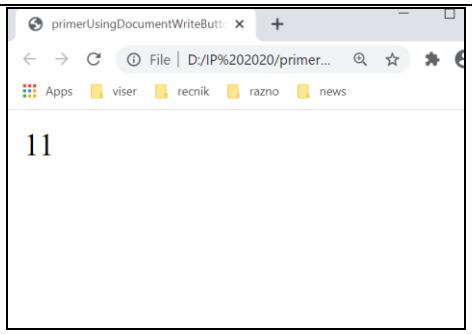


Слика 3.1.18 Уписивањем у *HTML* елемент коришћењем `document.write()`.

Начин извршавања је сличан као са `innerHTML`. Међутим, ако се користи у комбинацији са дугметом, након клика се брише све у прегледачи и остаје само резултат израчунавања.

```
<!DOCTYPE html>
<html>
  <body>
    <h2>Moja veb strana sa JS</h2>
    <p>Zbir brojeva 5 + 6 je</p>
    <h3 id="demo"></h3>
<button type="button" onclick="document.write(5+6)">Klikni</button>
  </body>
</html>
```





Слика 3.1.19 Уписивањем у *HTML* елемент коришћењем `document.write()` и `</button>`.

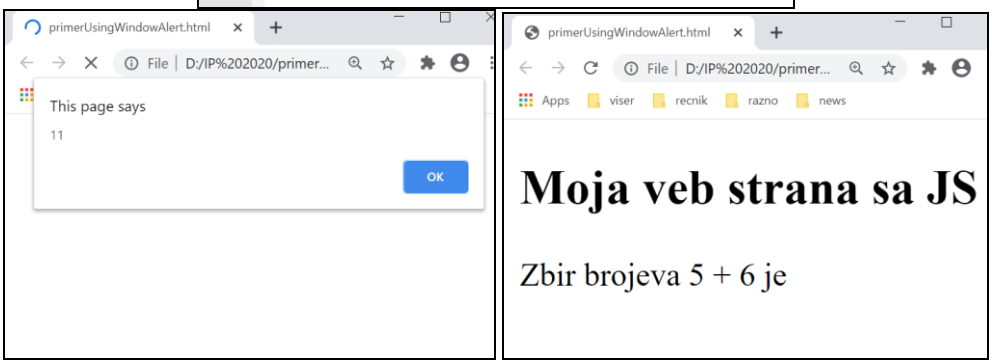
Слика 3.1.19 илуструје да се појављује само резултат 11 у прегледачу. Коришћење `document.write()` након што је *html* документ учитан може да обрише сав постојећи *html* садржај у прегледачу. Зато се метода `document.write()` користи само за потребе тестирања.

3.1.5.3 Коришћење *window.alert()*

Може да се користи прозор упозорења коришћењем `window.alert()`, да би се приказали подаци у прегледачу. На слици 3.1.20 илустрован је пример.

Када се учита *html* фајл, он најпре прикаже у посебном прозору то што је резултат скрипт фајла; након што се кликне на дугме, приказује се садржај који би иначе приказа *html* код у прегледачу.

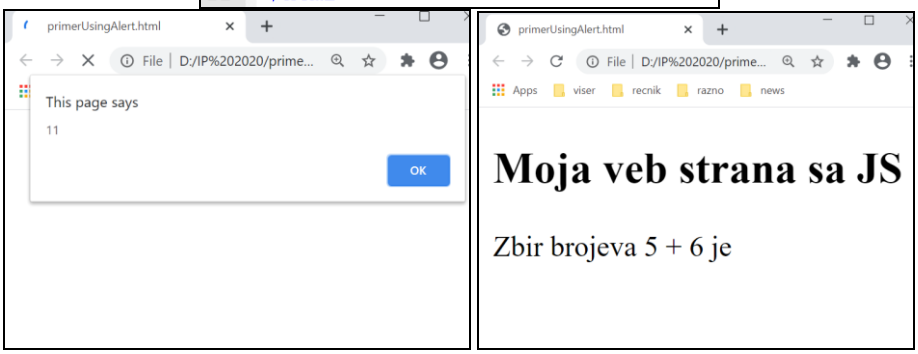
```
<!DOCTYPE html>
<html>
  <body>
    <h2>Moja veb strana sa JS</h2>
    <p>Zbir brojeva 5 + 6 je</p>
    <h3 id="demo"></h3>
    <script>
      window.alert(5+6);
    </script>
  </body>
</html>
```



Слика 3.1.20 Уписивањем у *HTML* елемент коришћењем `window.alert()`.

Једна од верзија јесте да се изостави назив `window`, као што је приказано на слици 3.1.21.

```
<!DOCTYPE html>
<html>
  <body>
    <h2>Moja veb strana sa JS</h2>
    <p>Zbir brojeva 5 + 6 je</p>
    <h3 id="demo"></h3>
    <script>
      alert(5+6);
    </script>
  </body>
</html>
```



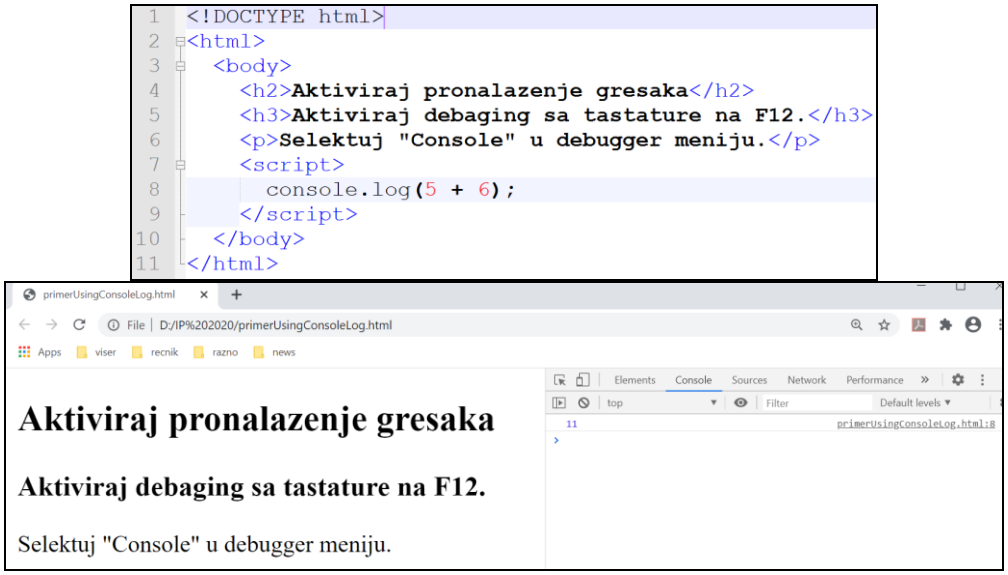
Слика 3.1.21 Уписивањем у *HTML* елемент коришћењем `alert()`.

У *JavaScript*-у, објекат `window` је глобални, што значи да има особине глобалне променљиве и да се подразумева да има све особине и методе које припадају објекту `window`. Стога је опционо спецификовање кључне речи *window*.

3.1.5.4 Коришћење *console.log()*

За потребе откривања грешака у програмском коду уобичајено је да се користи приказ резултата скрипт кода у конзоли. Користи се метода `console.log()`.

У примеру приказаном на слици 3.1.22 лево приказан је *html* документ који се види у прегледачу као на истој слици доле лево. Текст у прегледачу описује шта треба да се уради да би се видео приказ у конзоли.



Слика 3.1.22 Уписивањем у *HTML* елемент коришћењем `console.log()`.

Леви део слике се приказује када се отвори *html* фајл. Потребно је на тастатури притиснути функцијски тастер `F12`. Тада се појављује и десни део у прегледачу. Кликне се у десном делу на `Console`. Тада се појављује подебљана линија испод наслова за конзолу. Улога осталих наслова биће објашњена касније када буде описан рад за отклањање грешке.

У овом примеру, у едитору се види да у конзоли треба да се прикаже резултат збира два броја. У првој линији приказа се види број 11, што је у овом случају исправан резултат. У десном делу врсте у којој је резултат види се назив фајла, `primerUsingConsoleLog.html`, а после двотачке је број линије у којој је настао овај резултат; у линији 8 едитора се види да конзола треба да прикаже збир два броја. Резултат се не приказује у самом прегледачу; ако желимо да знамо неки међу-резултат који се не приказује у прегледачу, тада можемо на овај начин да проверимо да ли је резултат онај који треба да буде или је дошло до грешке, и зато програм не ради исправно. Наравно, брисањем линија 7, 8 и 9 у едитору неће се више појављивати резултат у конзоли али ни у прегледачу.

3.1.6 *JavaScript* наредбе

JavaScript програми се састоје од листе инструкција које рачунар може да изврши. У програмским језицима, ове програмске инструкције се називају наредбе (*statements*). У *HTML*-у, наредбе *JavaScript* програма се извршавају у прегледачу, без обзира да ли је *JavaScript* скрипт уграђен у *html* документ или се референцира на екстерни *JavaScript* фајл.

JavaScript наредбе се састоје од вредности (*values*), оператора (*operators*), израза (*expressions*), кључних речи (*keywords*) и коментара (*comments*).

На пример, извршавајући наредбу

```
document.getElementById("demo").innerHTML = "Zdravo Miro";
```

прегледач треба да напише `Zdravo Miro` унутар *html* елемента који има идентификатор `id="demo"`. Сличан пример је приказан на слици 3.1.8.

Већина *JavaScript* програма садржи већи број *JavaScript* наредби. Наредбе се извршавају редом како су написане у програму. У пракси се *JavaScript* програми или делови програма као скуп наредби називају *JavaScript* кодови, као што је у овом уџбенику често коришћено у претходним поглављима.

Ознака која се користи да би показала где је крај наредбе је тачка-зарез, ;. Ознака ; није обавезна да се означи крај наредбе, ако у линији у којој је наредбе не постоји ништа друго. Да би се избегла непријатна изненађења и грешка у програму, пожељно је да се користи ; за крај наредбе. На пример, ако следи друга наредба у наставку једне наредбе, и не постоји ознака ; тада ће се појавити грешка у извршавању. Ако се користи едитор за писање кода, дугачке наредбе из једне линије програма могу бити приказане у следећем реду, што програмеру може изгледати као да је у новој линији. На слици 3.1.23 је показан исправан начин рада када су три наредбе за доделу вредности променљивима у линијама 9, 10 и 12 у едитору лево, а испис резултата се исправно приказује у прегледачу десно.

Слика 3.1.23 Наредбе са и без ознаке ; за исправан рад у прегледачу.

Наредба за испис је у линији 12, али се она приказује и у реду испод линије 12, зато што едитор врши прелом дугачких линија кода на више редова.

На слици 3.1.24 је показан пример где је показано да се у линији кода 10, налазе две наредбе при чему после прве наредбе не постоји ознака ; а друга наредба је у истој линији.

Слика 3.1.24 Више наредбе без ознаке ; у једној линији кода не даје исправан резултат у прегледачу.

Визуелно код изгледа исто као на слици 3.1.23 али не приказује резултат `11` у прегледачу. Друга наредба се приказује у реду испод линије 10, зато што едитор аутоматски прелама дугачку линију у два реда па програмер има утисак да су то две линије кода.

Да би се избегли ови проблеми, програмер треба увек да напише знак ; на крају сваке наредбе, и програм ће исправно радити као на слици 3.1.25.

Слика 3.1.25 Више наредби са ознакама ; у једној линији кода дају исправан резултат у прегледачу.

У примеру са слике 3.1.25 није проблем што се у линији 10 у едитору појављује још једна наредба која је приказана у следећем реду због тога што едитор аутоматски прелама дугачке линије кода. Сада се исправно приказује резултат `11` у прегледачу као што се види на слици 3.1.25 десно.

3.1.6.1 Употреба бланко карактера

У примеру са слике 3.1.25 унет је велики број бланко карактера (белих размака са тастатуре када се притисне дугачки тастер у доњем реду тастатуре) између две наредбе `b = 6;` и `c = a + b;`. Програм исправно ради без обзира на унете бланко карактере. Ова особина се може искористити за писање прегледног и читљивог кода. Уместо да се наредбе куцају без размака, на пример

```
var osoba="Miro";
```

боље је да се унесу бланко карактери, на пример

```
var osoba = "Miro";
```

У оба случаја ће бити идентичан приказ у прегледачу. Треба обратити пажњу да уношење бланко карактера између наводника прегледач те бланко карактере третира као део речи и они ће се појавити и при приказу у прегледачу.

Добра је пракса да се користе бланко карактери и око оператора (`= + - * /`), на пример, ако су познате вредности за `y` и `z`

```
var x = 2 * y + z;
```

Још је боље да се користе и заграде да би се груписао редослед извршавања, на пример различити резултати се добијају у примени следеће две наредбе

```
var x = (2 * y) + z;
var x = 2 * (y + z);
```

У првој наредби се најпре израчуна резултат множења `2 * y` па се на тај резултат дода сабирак `z`. У другој наредби се најпре израчуна збир `(y + z)` па се тај резултат помножи са 2.

Већина програмера мисли да зна редослед извршавања, али се не ретко дешава да програм прикаже другачији резултат. То се посебно односи на операције дељења, када се мисли да ће све лево од знака за дељење `/` бити извршено, па онда тек операција дељења. Док је већина програмера сигурна у редослед аритметичких операција, мало је оних који са сигурношћу знају редослед логичких операција. Заграде имају виши приоритет од операција, па је пожељно да се групишу операције за које се сигурно жели првенство при израчунавању.

У сваком случају треба користити бланко карактере који чине код читљивијим уместо без размака, као што је пример исправног али мање читљивог кода

```
var x=2*y+z;
```

Један од разлога коришћења кондензованог писања (без бланко карактера) јесте дужина линије наредбе. За добар читљив код се препоручује да се избегавају линије кода дуже од 80 карактера. Ако није могуће, тада треба направити прелом на месту оператора, на пример доделе вредности `=`

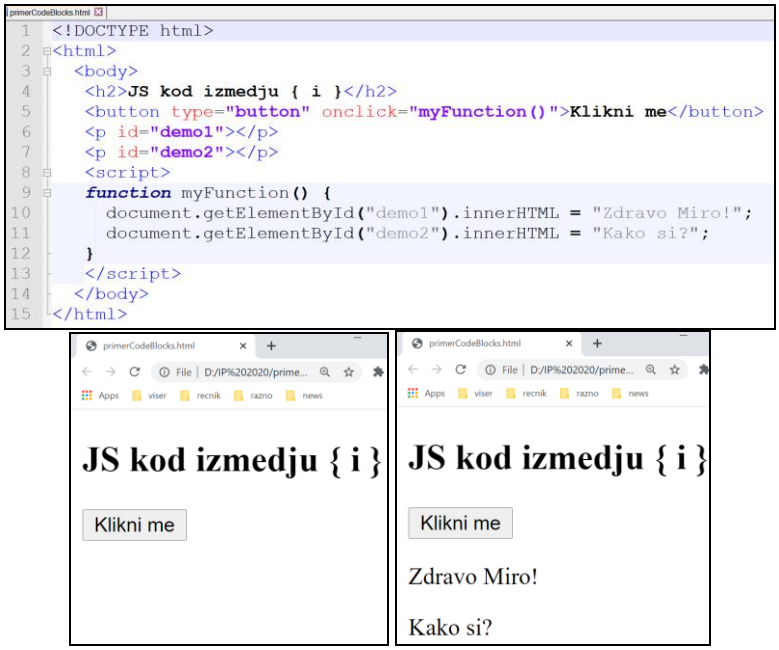
```
document.getElementById("demo").innerHTML =
```

```
        "Zdravo Miro!";
```

На слици 3.1.26 у линији 8 едитора дошло је до аутоматског прелома линије у следећи ред, али то није нова линија кода, већ нови ред линије број 8.

Слика 3.1.26 Аутоматски прелом и прелом код оператора.

У линији 13 је урађен прелом после оператора =, тако да је наставак линије са наредбом у линији 14. Овакав прелом кода је читљив и бољи зато што програмер контролише прелом линије уместо едитора. Важно је напоменути да прелом између наводника који треба да генеришу *html* код проузрокује грешку чак ако је и после оператора, као што је показано у једном од примера из претходних поглавља.



Слика 3.1.27 Блок кода између витичастих заграда { и }.

.

3.1.6.2 Употреба функција

У примеру са слике 3.1.27 унет је блок наредби преко функције.

Кликом на дугме се активира функција која има две наредбе за различите идентификаторе. Први идентификатор генерише текст за параграф после дугмета, а други идентификатор у функцији генерише текст за следећи параграф.

Као што се обичне заграде користе за груписање око оператора, тако се витичасте заграде користе за груписање блокова кода између витичастих заграда (*curly brackets*), {...}. Сврха је да се одреди приоритет извршавања групе наредби као и код оператора. Пример је урађен за функцију (*JavaScript functions*).

И у овом примеру су унети бланко карактери да би се увукле наредбе и да би код био читљивији.

3.1.6.3 Употреба кључних речи

У овим примерима коришћене су променљиве и назив функције које могу бити произвољне. Међутим, не могу се користити било које речи за називе променљивих. Постоје такозване кључне речи (*JavaScript Keywords*) које су резервисане за наредбе и које се не могу користити за називе променљивих или функција. *JavaScript* наредба често почиње са кључном речи која идентификује JavaScript активност која се извршава том наредбом. На сајтовима организација, које објављују туторијале, може се наћи списак резервисаних речи које се користе као кључне речи. У следећој табели је списак неких често коришћених речи:

Кључна реч (Keyword)	Опис
`break`	Прекида `switch` или петљу (`loop`)
`continue`	Излази из петље и стартује од почетка
`debugger`	Прекида извршавање *JavaScript* кода и позива (ако је могуће) *debugging* функцију
`do ... while`	Извршава блок наредби, и понавља извршавање, све док услов не добије вредност `true`
`for`	Означава блок наредби које се извршавају све дотле док је услов `true`
`function`	Декларише функцију
`if ... else`	Означава блок наредби које се извршавају у зависности од услова
`return`	Излази из функције
`switch`	Означава блок наредби које се извршавају у зависности од различитих случајева
`try ... catch`	Имплементира *error handling* на блок наредби
`var`	Декларише променљиву

У претходним примерима већ је коришћена кључна реч `var` када су дефинисане промењиве.

3.1.7 *JavaScript* синтакса

JavaScript синтакса је сет правила по којима се праве *JavaScript* програми. На пример, посматрајмо следеће три наредбе:

```
var x, y, z;
x = 5; y = 6;
z = x + y;
```

Наредба у првој линији кода декларише три промењиве `x, y, z`. Друга линија кода садржи две наредбе доделе вредности `x = 5;` и `y = 6;`. Трећа линија кода израчунава збир две промењиве чије вредности су додељене у претходном делу коду (`x + y`), и додељује резултат промењивој `z`. Важно је уочити да су са леве страна знака `=` симболички називи промењивих, а са десне стране знака `=` познате вредности, у овом случају бројеви `5` и `6`.

3.1.7.1 Типови вредности

JavaScript синтакса дефинише два типа вредности: непромењиве вредности и промењиве вредности. Непромењиве вредности се називају константама (*literals*). Промењиве вредности се називају промењивима. Бројеви се пишу са или без децималне тачке, за разлику од српског језика где се користи зарез да раздвоји целобројни и разломљени део броја. Стрингови су текст који се налази између двоструких или једноструких наводника.

У програмским језицима називи промењивих (на пример `x`) се користе да означе део простора у меморији рачунара у који ће бити смештене вредности промењивих. Након доделе вредности (`x = 5`) у тај меморијски простор се смешта бројна вредност (`5`). Програмер види симболичко име промењиве (`x`) али не зна, и не треба да се зна, где се тачно алоцира део меморије. Да би се заузео меморијски простор користи се кључна реч `var`, и ово се назива декларисање промењиве. Затим се наводи најмање један назив промењиве (`x`) или више промењивих (`x, y, z`). За сваку промењиву се резервише простор у који ће бити смештене вредности после доделе, на пример `x = 5;` и `y = 6;`. Тек након доделе вредности промењивама `x` и `y`, може да се израчуна збир и резултат да се додели трећој промењивој `z`.

За сложенија израчунавања, а у циљу писања компактног и читљивог кода, могу се користити изрази (*expression*) који једном наредбом комбинујући промењиве, константе и операторе могу да израчунају неку вредност. Израчунавање вредности израза се назива *evaluation*.

Осим бројева, вредности могу да буду и стрингови, на пример `"Miro"`, `" "`, `"Lu"`, а исти оператор `+` може да произведе различите резултате. На пример, у следеће две наредбе оператор `+` генерише различите резултате

```
"Miro" + " " + "Lu"
5 * 10 + 1
```

У првој наредби оператор + спаја три стринга у један, тако што између две речи убацује бланко карактер, а у другој наредби оператор + извршава операцију сабирања:

```
"Miro Lu"
51
```

Нису све наредбе извршне. Код који се налази после двоструког знака за косу црту, //, или између /* и */ се не извршавају и третирају се као коментари.

3.1.7.2 Идентификатори и називи

JavaScript идентификатори су називи који се користе у коду да именују променљива, кључне речи, функције и лабеле. Правила за давање назива су слична у различитим програмским језицима.

У *JavaScript*-у, први карактер мора да буде слово, знак подвучена црта (*underscore*, _) или знак за долар (*dollar sign*, $). Сви наредни карактери назива могу да буду слова, бројеви, знак подвучена црта или знак за долар. Бројеви не могу да буду први карактер у називу. На овај начин *JavaScript* може да разликује идентификаторе од бројева.

JavaScript прави разлику између малих и великих словних карактера, и каже се да је *Case Sensitive*. Сви JavaScript идентификатори су *case sensitive*. На пример, променљиве које имају називе `lastName` и `lastname` су две различите променљиве зато што је у првом називу велико слово `N` а у другом је мало слово `n`.

Исто правило важи и за кључне речи; *JavaScript* неће интерпретирати речи `VAR` или `Var` као да су кључна реч `var`. Уобичајено је правило да у програмским језицима у којима кључне речи почињу великим словима да називи променљивих и функција почињу малим словима, и обрнуто, ако већина кључних речи почиње малим словом, тада називи променљивих и функција треба да почињу малим словом. У овом уџбенику биће коришћена и мала и велика слова у ситуацијама када се са сигурношћу зна да не постоје сличне кључне речи које би могле да доведу до забуне и грешака у кодовању.

Начин писања једне речи која је састављена од две или више других речи тако да свака посебна реч почиње великим словом а сва остала слова су мала, назива се *Camel case*, *camelCase* или *CamelCase*. Управо је и назив програмског језика написан на овај начин *JavaScript*, указујући на назив *Java* у комбинацији са тиме да се ради о скриптном типу *Script*. Прво слово оваквог типа сложене речи може да буде мало или велико слово, у зависности од правила које сам програмер изабере по својој вољи или правилима која важе у програмерској фирми, на пример *dreamWorks* (тип *lower camel case*) или *DreamWorks* (тип *Pascal case*, *upper camel case*). Прво слово сложене речи може да се изабере да буде мало у случајевима када је прва реч само једно слово, на пример *iPhone* и *eBay*, Подразумева се да се за писање програма не користе само енглеска латинична слова.

Историјски могла је да се користи цртица, на пример *dream-works*, међутим тада би могло да се направи забуна да је то разлика две променљиве, као да је *dream - works*. Такође, могао је да се користи бланко карактер, *dream works*, али би друга решења могла да се интерпретира као аргумент или вредност параметра. Да би се избегла двозначност и код био читљивији, у пракси се у *JavaScript*-у користи *CamelCase*.

Иако се ради прегледности кода може користити и знак подвучена црта (_), није потребно проверавати број карактера у називу. Подразумева се да ће програмери користити *lower camel case*, на пример *dreamWorks*.

JavaScript користи *Unicode character set*, који покрива скоро све словне и бројне карактере, интерпункцијске знакове и симболе који се користе у пракси.

3.1.8 *JavaScript* коментари

JavaScript коментари се уобичајено користе да објасне код, тако да програм буде читљив и да буде лакше одржавање програма у случају потребе за изменама у коду. Током развоја програма, дешава се да програм не функционише према очекивањима. Тада програмер обично постави коментаре у линије кода које су могућ узрок грешке, и напише нове линије програма. Тестирањем дела кода, на пример исписивањем међурезултата, проверава се исправност дела програма. У следећем примеру је показано како се користе коментари у једној линији при чему коментар почиње са двоструким укошеним знаком `//` и *JavaScript* игнорише текст све до краја линије кода, односно *JavaScript* неће извршавати нити интерпретирати наредбу:

```
// Ispisivanje sadrzaja paragrafa:
document.getElementById("myP1").innerHTML = "Moj prvi paragraf";
// Testiranje sadrzaja paragrafa:
// document.getElementById("myP2").innerHTML = " Moj prvi test";
```

У овом примеру, прва линија је коментар шта треба да уради наредба која следи у следећој линије. Друга линија је сама наредба која би требала да се изврши, а на основу коментара се очекује исправан резултат. Трећа линија коментара сугерише да следи наредба која служи за тестирање. Четврта линија садржи команду која треба да изврши то што је најављено коментаром, али без `//`. Када се утврди да тест наредба исправно ради, напишу се линије за коментар `//` . При поновном извршавању, четврта линија која сада има на почетку `//`, неће извршавати ову наредбу. Тест линије не треба да се бришу зато што у било ком тренутку се може поновити тест извршавање наредбе из четврте линије кода једноставним брисањем двоструких косих црта.
У следећем примеру коментари су додати у наставку наредби у све три линије:

```
var x, y;       // Deklarise promenljive x i y
x = 5; y = 6;   // Dodeljuje vrednosti promenljivoj x i y
var z = x + y;  // Deklarise z i dodeljuje joj vrednost zbira
```

У свакој линији кода, наредба се извршава закључно са двоструком косом цртом, а од знака `//` па све до краја линије интерпретер игнорише текст.

У ситуацијама када је потребно да се коментарише већи број линија кода, као блок текста или више наредби, тада се цео блок линија смешта између `/*` и `*/`:

```
/*
Test primer sa deklaracijom, dodelom vrednosti promenljivoj,
Izracunavanja izraza za promenljive x, y i z

var x, y;       // Deklarise promenljive x i y
```

```
x = 5; y = 6;   // Dodeljuje vrednosti promenljivoj x i y
var z = x + y;  // Deklarise z i dodeljuje joj vrednost zbira
*/
```

Када се утврди да је део кода исправан, последњи ред који садржи `*/` се премести у бланко ред, тако да само два реда текста из овог примера остају коментарисана.
У енглеској терминологији овакав начин коментарисања дела текста и наредби се назива *multi-line comment* (*comment block*) и од посебног је значаја за документацију софтвера.

3.1.9 *JavaScript* променљиве

У поглављу о синтакси већ је дат пример наредби које се односе за декларисање променљивих и начин како се користе:

```
var x, y, z;
x = 5; y = 6;
z = x + y;
```

Суштина променљивих јесте да се одреде контејнери (простор у меморији) који ће садржати податке о променљивима, тако да се наредбама за доделу у те контејнере убацују вредности. Када је потребно, из контејнера се узимају вредности за израчунавања, тако да се израчунате вредности користе за смештање у исте или друге контејнере или се резултат приказује у складу са другим наредбама. Наредба у првој линији претходног кода декларише три променљиве `x, y, z`, чиме се резервишу симболичка имена преко којих се смештају вредности у контејнере или се исчитавају из контејнера. Друга линија кода садржи две наредбе доделе вредности `x = 5;` и `y = 6;` којима се у контејнере смештају вредности. Трећа линија кода чита вредности у контејнерима и користи их за израчунавање као збир две променљиве (`x + y`), а затим израчунату вредност додељује променљивој `z`.

У поглављу о *JavaScript* идентификаторима објашњена су правила о дефинисању назива променљивих. Иако смо у претходном примеру користили једнословне називе променљивих, то има смисла за потребе тестирања и концизног писања програма. У циљу одржавања софтвера, назив променљиве треба да буде такав да асоцира на сврху променљиве тако да свако ко проучава код може лако да уочи улогу те променљиве у програму. На пример, добри називи би били `proizvodnaCena`, `marzaZaOpremu`, `PdvZaOpremu`, `prodajnaCenaDiskont`, при чему се користи *CamelCase*. Ово је посебно важно да би се при писању израза одмах уочила грешка, на пример да ли две променљиве треба да се помноже (множење са 1.25 за додати ПДВ), или збир две цене за добијање укупне цене.

Све *JavaScript* променљиве морају да се идентификују својим јединственим називом, зато што меморијском контејнеру може да одговара само јеан симболички назив. У ситуацији када постоје слични називи, добра је пракса да се користе коментари непосредно пре наредби или у истој линији кода са наредбом, тако да увек може лако да се провери који назив променљиве одговара разлогу за кодовање. У поглављу о *JavaScript* идентификаторима објашњена су правила који карактери могу да се користе за називе променљивих и да *JavaScript* прави разлику између малих и великих словних карактера.

JavaScript оператор `=` има прецизно и недвосмислено значење. Са леве стране се налази назив резервисаног контејнера, а са десне стране вредност или израз којим се израчунава

шта треба да буде унето у контејнер. Сви чланови са десне стране морају да имају познату вредност јер ће у противном програм пријавити грешку. Важно је уочити да знак једнакости није наредба којом се каже да је лева страна једнака десној, већ да значи да то што је са десне стране се додељује промењливој која је са леве сране једнакости. Оваква интерпретација је у супротности за значењем знака једнакости у алгебри, где вредност са леве стране може да буде и број, а да се са десне стране налази назив промењиве. Следећи пример показује разлику између алгебарског значења знака једнакости и значења у *JavaScript*-у:

```
x = 5;
x = x + 6;
```

У првој линији је урађена додела промењливој `x`. У другој линији кода се преузима из меморијског контејнера вредност `5`, додаје се вредност `6`, израчунава збир који даје резултат `11`, и та вредност уписује у меморијски контејнер `x`. На овај начин је промењена вредност у контејнеру који има симболички назив `x`. Када би ове линије кода посматрали на алгебарски начин, испало би да је `5` једнако `11`, што је нетачно.

У *JavaScript*-у се користи ознака `==`, да означи оператор да је лева страна једнака десној. У претходном примеру када би једноструки знак једнакости био замењен са `==`, тада би *JavaScript* генерисао резултат да исказ није тачан.

3.1.9.1 Типови података

У овом делу биће описани различити начини рада са бројевима и стринговима. Да би се разликовали стрингови и бројеви, стрингови се пишу између двоструких или једноструких знакова за наводнике (нпр. `"Moj prvi paragraf"`), а бројеви се пишу без наводника (`3.14`). Број који је написан између наводника се третира као текстуални стринг (`"3.14"`).

Декларисање промењливе је први корак у креирању промењливе чиме се резервише контејнер у који ће бити смештене вредности. На пример, у следећим наредбама

```
var cenaAuto;
cenaAuto = "15000 EUR";
```

У првом реду се декларише промењлива са називом `cenaAuto`. Након декларације, не зна се која је вредност промењиве. Зато ће *JavaScript* интерпретирати да је вредност промењиве `undefined`, односно недефинисана.

У другој линије се додељује вредност; иако се очекује да цена буде број, у овом случају то је текстуални стринг који укључује и валуту у којој је изражена цена. Цена може да се изрази бројем, а валута да се додели новој промењливој која је сада декларисана у првој линији наредбе:

```
var cenaAuto, cenaValuta;
cenaAuto = 15000;
cenaValuta = "EUR";
```

У овом другом случају, промењиве заузимају мањи простор у меморији и одмах су припремљене за алгебарска израчунавања; ако има више цена, нема потребе да се валута придодаје као текстуални стринг броју који означава цену.

Добра је пракса да се све промењиве декларишу на почетку програма, тако да је једноставније да се пронађу у програму, а и програм ће се ефикасније извршавати. Наиме, сваки пут када се декларише променљива рачунар прво проналази у меморијском простору где постоји слободан простор, затим резервише (алоцира) тај простор за променљиву. Када се касније додељује вредност променљивој, рачунар зна где је резервисан простор за променљиву и врши само упис вредности.

Добра пракса је и да се одмах при декларисању променљиве изврши додела подразумеване вредности како не би била недефинисана вредност `undefined`. На пример, може да се користи једна наредба за више променљивих, при чему после сваке променљиве, или доделе вредности треба да се користи обичан зарез:

```
var cenaAuto = 15000, cenaValuta = "EUR";
```

Знак тачка-зарез, `;`, означава крај наредбе за декларисање свих променљивих. Ради прегледности, свака нова декларисана променљива може да буде у новом реду, тако да се наредба простире у више линија, с тим да се свака линија пре нове променљиве завршава зарезом:

```
var cenaAuto = 15000,
cenaValuta = "EUR";
```

Ако се након декларисања променљиве, и доделе вредности, поново декларише променљива са истим називом, вредност променљиве ће остати она која је последња додељена, што значи да вредност неће бити `undefined`.

При декларисању променљиве може са десне стране једнакости да буде и израз ако је број, или израз са стринговима ако је променљива стринг, на пример конкатенација два текстуална стринга.

За израчунавање израза, при декларисању променљиве, користе се иста правила као и за извршавање израза за аутоматску конверзију. На пример, ако је први члан израза број задат као стринг а други члан израза је број, а операнд је знак за сабирање, тада се други члан конвертује у стринг и оператор за сабирање извршава конкатенацију два стринга. Исто се дешава ако је други члан број као стринг.

Први карактер назива променљиве може да буде знак за долар или подвучена црта (`$`, `_`). Оба карактера се ретко користе у *JavaScript*-у, али имају своје значење када се користе у комбинацији са *JavaScript library jQuery*, или за означавање скривених (*hidden* и приватних) променљивих.

3.1.10 *JavaScript* оператори

JavaScript има већи број оператора који се могу распоредити у више група, као што је приказано у следећим табелама.

- Аритметички оператори,
- Оператори доделе,
- Оператори упоређивања
- Оператори са стринговима

Оператор	Опис аритметичких оператора
+	Сабирање (*addition*)
-	Одузимање (*subtraction*)
*	Множење (*multiplication*)
**	Експонент (*exponentiation*, ES2016)
/	Дељење (*division*)
%	Модуо, остатак дељења (*modulus, division remainder*)
++	Повећање (*increment*)
--	Смањење (*decrement*)

Оператор	Опис оператора доделе	
	Пример	Еквивалентан запис
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Оператор	Опис оператора упоређивања (релациони оператори)
==	Лева страна једнака десној страни (*equal to*)
===	Једнаке вредности и типови (*equal value and equal type*)
!=	Није једнако (*not equal*)
!==	Није једнако по вредности или типу (*not equal value or not equal type*)
>	Лева страна већа од десне (*greater than*)
<	Лева страна мања од десне (*less than*)
>=	Веће или једнако (*greater than or equal to*)
<=	Мање или једнако (*less than or equal to*)
?	Тројни (тернарни) оператор (*ternary operator*)

Ако *JavaScript* оператор има један операнд он се назива унарни оператор. Ако оператор има два операнда, назива се бинарни оператор. Оператор који има три операнда назива се тернарни или тројни оператор.

3.1.10.1 *JavaScript* стринг оператори

Неки оператори важе само за бројеве док други оператори дају другачији резултат ако се користе промењиве типа стринг, На пример, оператор + генерише различите резултате ако је један операнд промењива типа стринг

```
var ime = "Miro";
var blankoZnak = " ";
var prezime = "Lu";
var imeOsobe = ime + blankoZnak + prezime;
```

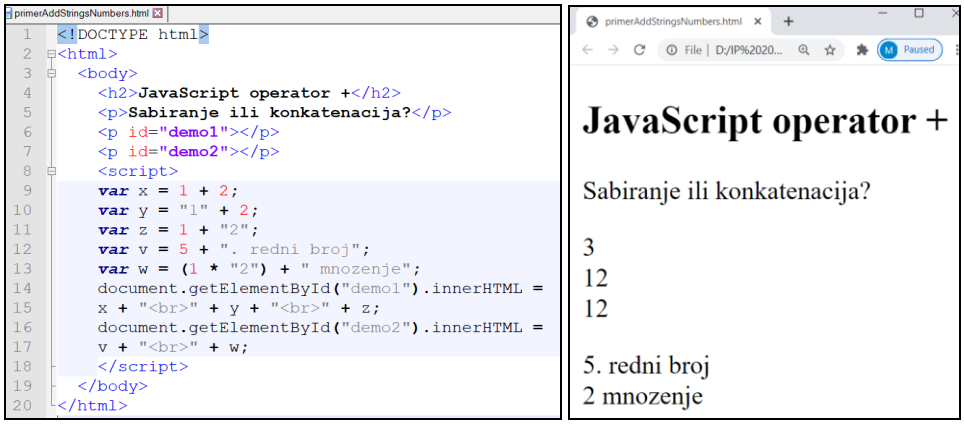
када се добија да је резултат `imeOsobe` конкатенација три стринга, `"Miro Lu"`.

Када су промењиве по типу бројеви

```
var cena = 80;
var marza = 10;
var pdv = 20;
var ukupnaCena = cena + marza + pdv;
```

добија се резултат `ukupnaCena` као збир три броја, 110.

На слици 3.1.28 приказано је неколико примера декларисања промењивих са доделом различитих типова промењивих.



Слика 3.1.28 Оператор + за аритметичко сабирање и конкатенацију стрингова.

При првом декларисању, када су оба сабирка бројеви, оператор + израчунава аритметички збир сабирака, па је резултат 3. При другом и трећем декларисању промењивих, први или други операнд је стринг, па се операнд који је број конвертује у стринг, а затим резлтат добија конкатенацијом (спајањем) два стринг; резултат је стринг `"12"`. Код четвртог декларисања први операнд, који је број, се конвертује у стринг и затим извршава конкатенација два стринга. У последњој декларацији, прво се конвертује чиниоц који је 2 типа стринг у тип број, извршава се операција множења у загради као операција вишег приоритета, затим се број добијен множењем конвертује у стринг и спаја са трећим операндом који је стринг. Због аутоматских конверзија типова података, који могу да дају другачији резултат од оног који очекује програмер, добра је пракса да се групишу у заграде оне операције које имају виши приоритет.

При приказу резултата у прегледачу, на основу онога што се види, не може се идентификовати да ли је резултат типа број или типа стринг, ако су резултат састоји само од цифара. У таквим случајевима је од посебног значаја, при тестирању рада програма, утврдити који је тип променљиве. За ту сврху се могу користити оператори типа из следеће табеле.

Оператор	Опис оператора типа
`typeof`	Враћа тип променљиве
`instanceof`	Враћа `true` ако је инстанца типа објекат (*object type*)

3.1.10.2 *JavaScript* логички оператори и оператори на нивоу бита

Логички оператори се често користе да се утврди тачност логичког израза, и на основу логичке вредности, која може да буде `true` или `false`, може доћи до гранања програма како би се извршио другачији блок наредби. У следећој табели су дат најважнији логички оператори.

Оператор	Опис логичког оператора
&&	Логичко И (*logical and*)
||	Логичко ИЛИ (*logical or*)
!	Логичка негација (*logical not*)

У многим апликацијама се користе такозвани бит-оператори (*bitwise* оператори) који оперишу на нивоу бита. Опис и примери примене ових оператора су дати у следећој табели.

Оператор	Опис оператора над битовима				
	Опис	Пример	Исто као	Резултат	Децимална вредност
&	AND	`5 & 1`	`0101 & 0001`	`0001`	1
|	OR	`5 | 1`	`0101 | 0001`	`0101`	5
~	NOT	`~ 5`	`~0101`	`1010`	10
^	XOR	`5 ^ 1`	`0101 ^ 0001`	`0100`	4
<<	Zero fill left shift	`5 << 1`	`0101 << 1`	`1010`	10
>>	Signed right shift	`5 >> 1`	`0101 >> 1`	`0010`	2
>>>	Zero fill right shift	`5 >>> 1`	`0101 >>> 1`	`0010`	2

Бит-оператори раде као бројеви представљени са 32 бита. Било који операнд се најпре конвертује у 32-битни број. Резултат се конвертује у *JavaScript* број.

У претходним примерима у табели, коришћени су 4-воро-битни бројеви без знака (*unsigned bits*), односно сви бројеви су позитивни. *JavaScript* користи 32-битне бројеве са знаком (знак одређује да је број позитиван или негативан). Због тога негација бита на пример у *JavaScript*-у, `~ 5` неће вратити резултат 10, већ ће резултат бити -6. Написано битовима, израз за негацију броја 5 је лево, а резултат је десно:

`~00000000000000000000000000000101` резлтат је `11111111111111111111111111111010`.

3.1.11 *JavaScript* аритметика

Аритметика је грана математике која проучава основне особине операција са бројевима. Осим аритметике, у модерној математици се појављују алгебра, геометрија и анализа. Основне операције су сабирање, одузимање, множење и дељење. Најједноставнији калкулатори користе и операције квадрирања и кореновања. У програмирању се користе основне особине аритметике укључујући и првенство операција множење и дељење у односу на сабирање и одузимање. Груписање аритметичких израза у заграде обезбеђује да редослед израчунавања претходи другим израчунавањима у сложеним изразима као део наредби.

3.1.11.1 *JavaScript* аритметички оператори

Аритметички оператори се користе за израчунавања израза са бројевима, без обзира да ли се додељују декларисаним променљивима или константама. Основни оператори су описани у претходном поглављу 3.1.10. Чланови израза на којима се извршавају операције се називају операнди. Операнд може да буде константа или променљива, при чему се подразумева да променљива има додељену вредност, како резултат аритметичке операције не би био неодређен. У сложеним изразима се може појавити и више од једног оператора и више од два операнда. Да би се избегла двосмисленост израза и грешке при израчунавању, добра је пракса да се користе заграде за груписање израза који имају првенство при редоследу извршавања.

Значење и примена аритметичких оператора је слично као и у осталим програмским језицима, па се у овом поглављу неће разматрати. Уместо оператора `**` може да се користи `Math.pow(x,y)`, где су `x` и `y` операнди.

Неки оператори имају другачије значење у *JavaScript*–у у односу на математичку аритметику; на пример, двоструки знак за сабирање `++` увећава вредност променљиве за 1 (*increment operator*), а двоструки знак за одузимање `--` значи да се вредност смањује за 1 (*decrement operator*). Означавање оператора за инкремент и декремент може да буде испред и иза променљиве, што може да доведе до погрешног интерпретирања програмера. Зато програмер увек мора да има на уму да се аритметички оператори како их дефинише математика могу разликовати од аритметичких оператора како их интерпретира *JavaScript*.

3.1.11.2 Вредност првенства оператора

Вредност првенства оператора може се исказати бројем, тако да већи број оператора значи да има има виши приоритет у односу на оператор који има ниже приоритет. У следећим табелама су дате вредности приоритета за операторе. У случају када се појављују оператори исте вредности, извршавање је са лева у десно.

Приоритет	Оператор	Опис	Пример
20	( )	Груписање израза	(3 + 6)

Приоритет	Оператор	Опис	Пример
19	.	Раздвајање имена објекта и особине објекта (*member*)	osoba.ime
19	[ ]	Раздвајање имена објекта и особине објекта (*member*)	osoba["ime"]
19	()	Позив функције	myFunction()
19	new	Креирање објекта	new Date()

Приоритет	Оператор	Опис	Пример
17	++	Постфиксни инкремент (*Postfix Increment*)	i++
17	--	Постфиксни декремент (*Postfix Decrement*)	i--

Приоритет	Оператор	Опис	Пример
16	++	Префиксна инкремент (*Prefix Increment*)	++i
16	--	Префиксна декремент (*Prefix Decrement*)	--i
16	!	Логичко не	!(x==y)
16	typeof	Тип	typeof x

Приоритет	Оператор	Опис	Пример
15	**	Експонент (*ES*2016)	10**3

Приоритет	Оператор	Опис	Пример
14	*	Множење	10 * 3
14	/	Дељење	10 / 2
14	%	Остатак дељења	11 % 5

Приоритет	Оператор	Опис	Пример
13	+	Множење	7 + 3
13	-	Дељење	11 - 1

Приоритет	Оператор	Опис	Пример
12	<<	Померај у лево (*Shift left*)	x << 2
12	>>	Померај у десно (*Shift right*)	x >> 2
12	>>>	*Shift right* (*unsigned*)	x >>> 2

Приоритет	Оператор	Опис	Пример
11	<	Мање од	x < y
11	<=	Једнако или мање од	x <= y
11	>	Веће од	x > y
11	>=	Једнако или веће од	x >= y

Приоритет	Оператор	Опис	Пример
11	in	Особина објекта, *ECMAScript* 2015 (*ES*6) или касније	"PI" in Math
11	Instanceof	Инстанца објекта *ECMAScript* 2015 (*ES*6) или касније	instanceof Array

Приоритет	Оператор	Опис	Пример
10	==	Једнако	x == y
10	===	Стриктно једнако	x === y
10	!=	Неједнако	x != y
10	!==	Стриктно неједнако	x !== y

Приоритет	Оператор	Опис	Пример
9	&	И на нивоу бита (*Bitwise* AND)	x & y
8	^	Ексклузивно ИЛИ на нивоу бита (*Bitwise* XOR)	x ^ y
7	|	ИЛИ на нивоу бита (*Bitwise* OR)	x | y
6	&&	Логичко И (*Logical* AND)	x && y
5	||	Логичко ИЛИ (*Logical* OR)	x || y
4	? :	Условно гранање (*Condition*)	? "Yes" : "No"

Приоритет	Оператор	Опис	Пример
3	+=	Додела	x += y
3	/=	Додела	x /= y
3	-=	Додела	x -= y
3	*=	Додела	x *= y
3	%=	Додела	x %= y
3	<<=	Додела	x <<= y
3	>>=	Додела	x >>= y
3	>>>=	Додела	x >>>= y
3	&=	Додела	x &= y
3	^=	Додела	x ^= y
3	|=	Додела	x |= y

Приоритет	Оператор	Опис	Пример
2	yield	Паузира извршавање генератор функције (*Pause Generator Function*)	yield x
1	,	Зарез	5 , 6

3.1.12 *JavaScript* оператори доделе

JavaScript оператори доделе и еквивалентан запис су приказани у следећој табели.

Оператор	Опис оператора доделе	
	Пример	Еквивалентан запис
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y
&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
|=	x |= y	x = x | y
**=	x **= y	x = x ** y, (*ES7*)

Оператори доделе који су записани са најмање још једним карактером, осим знака =, извршавају извршавају још једну операцију која је описана у колони за еквивалентан запис.

3.1.13 Типови података у *JavaScript*-у

JavaScript променљиве могу да буду различитог типа, као на пример, бројеви, стрингови или објекти

```
var godina1 = 16;                                  // Number
var ime1 = "Miro";                                 // String
var objekat1 = {ime2:"Miro", prezime2:"Lu"};       // Object
```

У програмирању, посебно се изучавају структуре података, па је од посебне важности у сваком програмском језику да се зна које типове података подржава програмски језик. За различите типове података се примењују посебна правила што је од посебног значаја када се у неком изразу појављују различити типови података као на пример:

```
var ime3 = 16 + "Miro";
```

У циљу избегавања прекида извршавања програма, пријављивања грешака због коришћења различитих типова података, многи програмски језици имају посебна правила која производе резултат као очекивану интерпретацију израза. За овај пример, *JavaScript* ће интерпретирати као да је програмер написао следеће:

```
var ime3 = "16" + "Miro";
```

Када се у *JavaScript*–у примени оператор +, а један операнд је по типу број, док је други стринг, Број ће најпре бити конвертован у тип стринг, а оператор + ће извршити конкатенацију (спајање) два стринга у један стринг.

У претходном поглављу су приказани приоритети израчунавања, па је тако за оператор + приоритет 13. Ако се у истом изразу користи исти оператор (или оператори истог приоритета) тада се израчунавање извршава са лева у десно. Посматрајмо пример наредбе у којој постоје два оператора +

```
var ime3 = 4 + 16 + "Miro";
```

Како је приоритет са лева у десно, први оператор + има два операнда типа број па ће најпре израчунати збир два броја, који је 20. Други оператор + има са леве стране израчунату вредност типа број који је 20, а са десне стринг "Miro". Сада се примењује друго правило, конвертује се број 20 у стринг "20" и врши конкатенација стринга "20" и стринга "Miro". Коначна вредност наредбе за доделу променљивој ime3 је "20Miro".

Посматрајмо још један пример наредбе у којој постоје два оператора +

```
var ime3 = "Miro" + 4 + 16;
```

Како је приоритет са лева у десно, први оператор + има два операнда, први је типа стринг "Miro", а други типа број 4. У овом случају ће најпре бити промењен тип броја у тип стринг, па је други операнд сада "20". Оператор + врши конкатенацију два стринг и добија се вредност "Miro4". Други оператор +, има са леве стране тип променљиве стринг, "Miro4", а са десне стране је тип број, 16. По правилима за оператор +, најпре се

број конвертује у стринг `"16"`, а затим извршава конкатенација два стринга, тако да је коначан резултат стринг `"Miro416"`.

У овим примерима је показано да редослед извршавања може да произведе различите резултате чак и када се ради о операторима истог приоритета ако се разликују типови променљивих, у једном случају то је био стринг `"20Miro"` а у другом случају `"Miro416"`.

Употребом заграда које имају виши приоритет, најпре се могу сабрати други и трећи члан и наредби, када се добија број 20, па ће коначан резултат бити стринг `"Miro20"`, на пример:

```
var ime3 = "Miro" + (4 + 16);
```

Ако наредба има више од два операнда, тиме се скраћује писање кода, међутим то може довести и до другачијег резултата од оног како је програмер мислио да ће програм да ради.

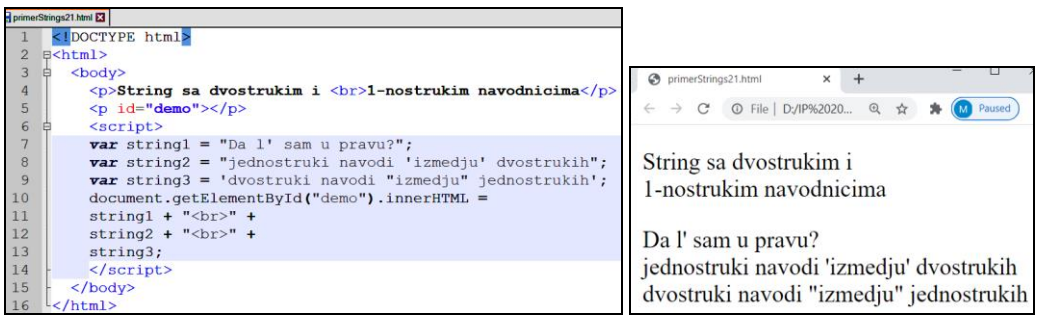
Декларисањем променљиве се врши резервација симболичког назива променљиве, али се у *JavaScript*-у не одређује тип променљиве. Стога се након декларисања променљивој може доделити било који тип података, на пример

```
var x;       // x je deklarisana i ima vrednost undefined
x = 5;       // promenljivoj x je dodeljena vrednost tipa broj
x = "Miro";  // promenljivoj x je dodeljena vrednost tipa string
```

Стринг је тип података који је секвенца карактера која се налази између једноструких или двоструких наводника. Следећа два израза генеришу исти стринг, дају исти резултат:

```
var string1 = "Miro Lu";   // koriscenje dvostrukih navodnika
var string2 = 'Miro Lu';   // koriscenje jednostrukih navodnika
```

На слици 3.1.29 је показано да је приказ идентичан ако се користе једноструки или двоструки наводници.



Слика 3.1.29 Стрингови са наводницима ' и ".

Ако стринг треба да садржи један тип наводника, на пример једноструке, тада се користи други тип наводника између којих је цео стринг, као што се види у прегледачу на слици 3.1.29.

JavaScript има само један тип бројева. И целобројни и рационални бројеви припадају истом типу, на пример:

```
var x1 = 34.00;       // Sa decimalnom tačkom
var x2 = 34;          // kao celobrojni
var y = 123e5;        // 12300000
var y = 123 * 100000; // 12300000
var z = 123e-5;       // 0.00123
var z = 123 / 100000; // 0.00123
```

Број цифара који се приказује углавном је мали због прегледности доделе. Велики бројеви могу да се напишу као производ са великим бројем нула (`* 100000`); уместо великог броја цифара, боље је да се користи експоненцијални запис где слово `e` означава експонент за основу 10; на пример, ознака `e5` означава 10^5, или са свим цифрама `* 100000`. Мали бројеви могу да се напишу као количник броја великог броја нула (`/ 100000`); уместо великог броја цифара, боље је да се користи експоненцијални запис где слово `e` означава експонент за основу 10; на пример, ознака `e-5` означава 10^{-5}, или са свим цифрама `/ 100000`.

Рационални бројеви се приказују са децималном тачком `34.0`, или у експоненцијалном облику са негативним експонентом `123e-5`, што је исто као да је написано `0.00123`.

Логички тип променљивих има две вредности, `true` (тачно) или `false` (нетачно). Често се ове вредности приказују и бројевима, 1 за `true` и 0 за `false`. На пример:

```
var x = 5;
var y = 5;
var z = 6;
(x == y);      // rezultat je true
(x == z);      // rezultat je false
```

Када се у изразима појављују различити типови података, *JavaScript* аутоматски мења у одговарајући тип пре него што се изврши израчунавање, као што је раније показано у примерима када је један операнд типа број а други операнд типа стринг.

JavaScript низови се записују у угластим заградама. Елементи низова су раздвојени зарезима.

Следећи пример декларише низ који има назив `prijatelji` и садржи три стринга са именима пријатеља, `Miro`, `Ana`, `Branka`:

```
var prijatelji = ["Miro", "Ana", "Branka"];
```

Првом елементу низа се придружује индекс 0 који се означава са [0], другом се придружује индекс 1 који се означава са [1], и тако даље, сваки наредни има индекс већи за 1 од претходног.

JavaScript објекти се пишу са витичастим заградама `{}`. Особине објеката се пишу као парови назив:вредност, раздвојени зарезима. На пример:

```
var osoba = {ime:"Miro", prezime:"Lu", godiste:50, ociBoja:"zelena"};
```

Објекат који има назив `osoba` у овом примеру има 4 особине: `ime`, `prezime`, `godiste`, `ociBoja`. Вредности су у овом примеру типа стринг и број.

Оператор `typeof` може да се користи да се одреди тип *JavaScript* променљиве. тако што враћа тип променљиве или израз. На пример за вредности типа стринг враћа резултат `"string"`:

```
typeof ""          // vraca rezultat "string"
typeof "Miro"      // vraca rezultat "string"
typeof "Miro Lu"   // vraca rezultat "string"
```

Када су вредности бројеви, враћа резултат `"number"`

```
typeof 0           // vraca rezultat "number"
typeof 231         // vraca rezultat "number"
typeof 3.14        // vraca rezultat "number"
typeof (8)         // vraca rezultat "number"
typeof (3 + 5)     // vraca rezultat "number"
```

Када је променљива декларисана, а није јој додељена вредност, тада и `typeof` враћа резултат `"undefined"`

```
var kosaBoja;    // vrednost je undefined, pa je i tip "undefined"
```

Променљивој може бити избрисана вредност доделом вредности `undefined`, и тада `typeof` враћа резултат `"undefined"`

```
kosaBoja = undefined; // vrednost je undefined, pa je i tip "undefined"
```

Када је променљива празна (`undefined`), са њом не може ништа да се ради. Међутим, стринг који је празан (нема ни једног карактера између знакова навода који декларишу променљиву типа стринг) и даље је типа стринг:

```
var string1 = ""; // vrednost je "", pa je tip "string"
```

У *JavaScript*-у `null` значи "ништа". То би могло да се интерпретира да је нешто што не постоји. Међутим, у *JavaScript*-у, тип података који је `null` у ствари за `typeof` враћа резултат да је тип објекат.

Ако је декларисан објекат, па му је додељена вредност `null`, тада он и даље остаје објекат.

```
var osoba = {ime:"Miro", prezime:"Lu", godiste:50, ociBoja:"zelena"};
osoba = null;    // vrednost je null, ali je tip objekat
```

Ако је декларисан објекат, па му је додељена вредност `undefined`, тада `typeof` враћа `undefined`.

```
var osoba = {ime:"Miro", prezime:"Lu", godiste:50, ociBoja:"zelena"};
osoba = undefined;   // vrednost je undefined i tip je "undefined"
```

Разлика између `undefined` и `null` може се најјасније разумети ако се погледа следећи пример из кога произилази да је за `null` вредност недефинисана али да је тип податка објекат:

```
typeof undefined     // undefined
typeof null          // object
null === undefined   // false
null == undefined    // true
```

Примитивне вредности података су оне које немају додатне особине и методе, и за које `typeof` оператор враћа неке од следећих примитивних типова:

- ✓ `string`
- ✓ `number`
- ✓ `boolean`
- ✓ `undefined`

На пример:

```
typeof "Miro" // vraca rezultat "string"
typeof 2.78   // vraca rezultat "number"
typeof true   // vraca rezultat "boolean"
typeof false  // vraca rezultat "boolean"
typeof x  // vraca rezultat "undefined" (ako x nema dodelu vrednosti)
```

Сложени типови података су они који за `typeof` оператор враћају неке од следећих типова:

- ✓ `function`
- ✓ `object`

На пример:

```
typeof {ime:'Miro', god:34}  // vraca rezultat "object"
typeof [1,2,3,4]             // vraca rezultat "object" (iako je "array")
typeof null                  // vraca rezultat "object"
typeof function myFunc(){}   // vraca rezultat "function"
```

Оператор `typeof` враћа `"object"` за низ зато што се у *JavaScript*-у низ третира као објекат.

3.1.14 *JavaScript* функције

JavaScript функције су блок наредби написаних тако да извршавају одређени задатак. *JavaScript* функција се извршава када се позове у програмском коду.

JavaScript функција се дефинише коришћењем кључне речи *function* после које следи назив функције и паром обичних заграда (). Назив функције се састоји од слова енглеске азбуке, бројева, или знака за подвучено и знака за долар, на исти начин како је то објашњено за називе променљивих. Између отворене и затворене обичне заграде могу да се налазе параметри раздвојени зарезима, на пример:

```
(parametar1, parametar2, ...)
```

Програмски код који треба да изврши функција се смешта између витичастих заграда, као у примеру

```
function nazivFunkcije(parametar1, parametar2, parametar3) {
  // kod i naredbe koje treba da budu izvrsene
}
```

Између обичних заграда, раздвојени зарезима, могу да се налазе називи параметара који се у блоку наредби користе као називи променљивих при дефинисању функције. При позиву функције која треба да се изврши, у заградама се појављују вредности и тада се ове вредности називају аргументима. Према томе, аргументи функције су вредности које се

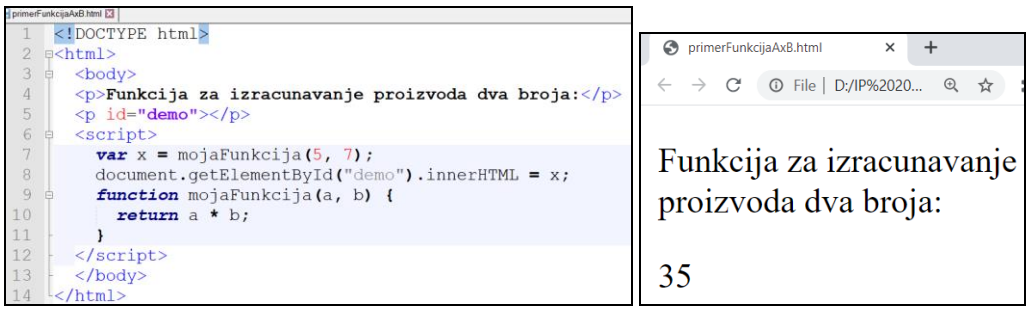
додељују функцији у тренутку позива функције, и блок наредби који је између витичастих заграда користи ове вредности да би произвео резултат који је дефинисан функцијом. Сви називи променљивих који се налазе у блоку наредби између витичастих заграда су локалне променљиве, којима се преносе вредности (аргументи) према називима параметара. То конкретно значи да називи параметара при дефинисању функције (који се налазе између обичних заграда) нису декларације променљивих са истим називима. У неким програмским језицима овако дефинисане функције се називају процедурама и подпрограмима.

Каже се да се функција позива. када се напише назив функције са вредностима аргумената које су између обичних заграда. Тада се наредбе које су између витичастих заграда извршавају тако што се називи променљивих у коду замењују вредностима аргумената које одговарају називима параметара функције.

Функција се позива (1) када се догоди неки догађаја (*event*), на пример кликне неко дугме, (2) када се позове из *JavaScript* програмског кода, или (3) се аутоматски позове (*automatically, self invoked*).

Након позива функције, извршава се блок наредби између витичастих заграда, а када се све изврши долази до прекидања извршавања и *JavaScript* функција враћа резултат на месту на коме је позвана. Ако се *JavaScript* функција позове из наредбе, *JavaScript* ће вратити (*return*) резултат после позивање наредбе. Најчешће се функција дефинише да би се израчунао неки резултат, па се зато враћена вредност (*returned value*)) враћа назад тамо где је функција позвана да се изврши (*back to the caller*).

У примеру са слике 3.1.30, у петој линији кода се позива скрипт код који се налази од од линије 6 до 12.

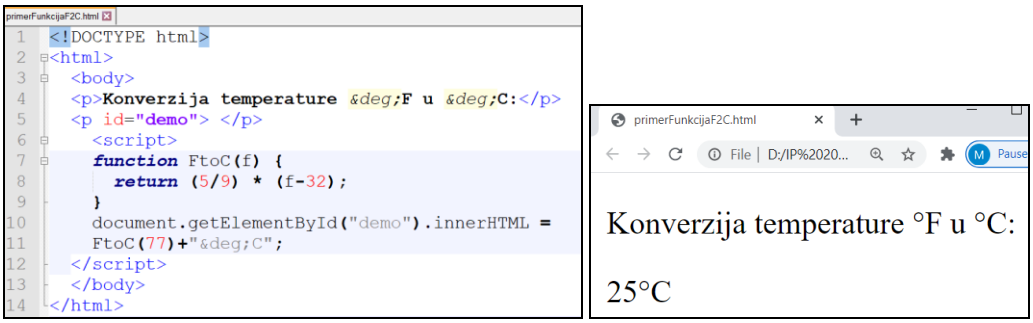


Слика 3.1.30 Функција која израчунава производ два броја.

У првој линији скрипт кода (линија број 7 на слици лево) налази се наредба за декларисање променљиве `x` и додела вредности коју израчунава функција `mojaFunkcija()`. Аргументи функције су бројеви `5` и `7`. *JavaScript* проналази дефиницију функције у линијама кода од 9 до 11. По дефиницији функције се утврђује да функција `mojaFunkcija` има два параметра. Првом параметру `a` се додељује први аргумент а то је вредност `5`. Другом параметру `b` се додељује други аргумент, вредност `7`. Функција извршава наредбу у линији 10, производ две променљиве које имају исте називе као и параметри функције, (`a` и `b`), Кључна реч `return` означава крај извршавања функције и израчуната вредност 35 се додељује променљивој `x` у линије кода 7. Ова вредност (35) се

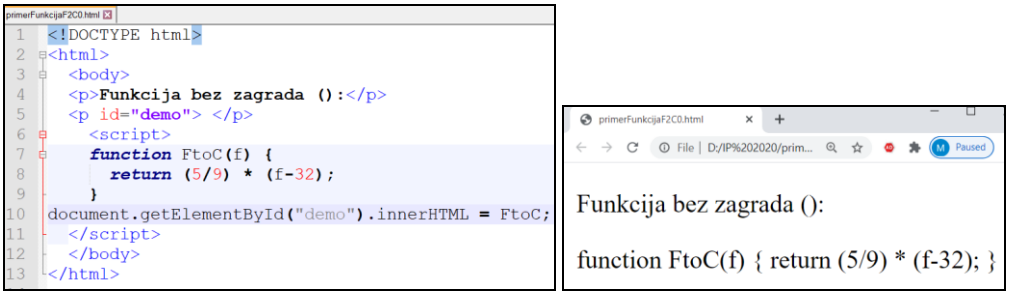
приказује у новом параграфу на месту где је позвана функција у прегледачу као што је приказано на слици 3.1.30 десно.

Иако на први поглед изгледа да је овакав начин кодовања компликован, зато што се овде ради о једноставном израчунавању, предности коришћења функције ће постати јасније када се буду анализирали сложенији примери. Овако једном написан код се може користити велики број пута, без потребе да се вишеструко тестира исправност кода. Код који је тестиран у алгоритамском смислу не треба поново да се тестира већ је довољно да се функцији прослеђују различити аргументи (на пример типа бројеви), функција ће израчунати нову другачију вредност у зависности од додељених вредности аргумената.



Слика 3.1.31 Функција за конверзију температуре из °F у °C.

На слици 3.1.31 приказан је код за конверзију температуре исказану у °F у температуру у °C. У линији број 5, у параграфу треба да се напише текст који је идентификован атрибутом као `id="demo"`. Када прегледач треба да интерпретира 5. линију кода, он тражи где се налази овакав идентификатор и проналази га у скрипт делу у линији 10. Линије 10 и 11 садрже код за генерисање текста, а за шта је потребно да се израчуна вредност функције. Назив функције је `FtoC()`, и функција има један параметар `f`. У 11 линији кода се види да је вредност параметра 77, који се као аргумент прослеђује функцији. Функција у оквиру своје дефиниције израчунава `(5/9) * (f-32)`, замењујући у линији 8 вредност променљиве `f` са атрибутом типа број који је `77`, израчунава аритметички израз и добијену вредност 25 линији кода број 11. Наредба из линије 11 (`FtoC(77)+"&deg;C"`) врши додатно израчунавање, па тако сабира број 25 са стрингом који је после знака `+,` (`"&deg;C"`). Како није могуће сабрати број и стринг, *JavaScript* врши конверзију броја 25 у стринг(`"25"`), и затим извршава конкатенацију два стринга (`"25"` и `"&deg;C"`) у нови стринг (`"25&deg;C"`). У прегледачу се приказује број 25, у наставку се приказује знак за степен `°` коришћењем `"&deg;"` а затим велико слово `C` мерне јединице за температуру у Целзијусовим степенима °C. Из овог примера је сада јасно да би директно коришћење кода сваки пут када је потребна конверзија из температуре у Фаренхајтовим степенима °F у Целзијусове степене °C значајно повећала број линија кода и учинила програм мање читљивим.



Слика 3.1.32 Функција позвана без заграда ().

На слици 3.1.32 приказан је пример у коме се функција позива без обичних заграда. Прегледач не извршава код, и не приказује резултат после извршавања функције, већ приказује функцију као објекат и приказује дефиницију функције.

У *JavaScript* –у, функције могу да се користе на исти начин као и променљиве, у свим типовима додела, формула и израчунавањима. На пример, уместо да се користи променљива којој ће бити додељена вредност као резултат враћен извршавањем функције,

```
var x = FtoC(77);
var text = "Temperatura je " + x + " &deg;C"
```

функција може да се користи директно

```
var text = " Temperatura je " + FtoC(77) + " &deg;C";
```

као да је вредност коју враћа функције у ствари вредност променљиве, на сличан начин како је показано на слици 3.1.31.

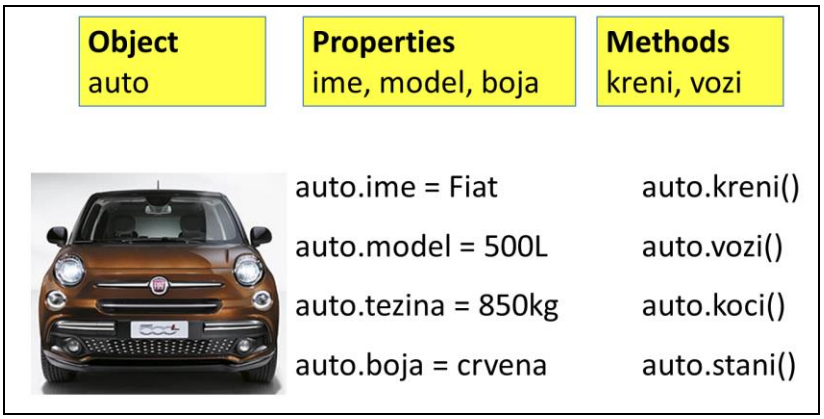
Променљиве декларисане у функцијама су локалне променљиве). Локалној променљивој може да се приступи само унутар функције. То значи да променљива изван функције која има исти назив као и променљива унутар функције неће променити вредност ако се истоименој променљивој у функцији промени вредност. Променљива у коду и истоимена променљива декларисана у функцији су две различите променљиве. Локална променљива се креира када се започне извршавање функције, користи се за време извршавања функције, и брише се када функција заврши са извршавањем функцијских наредби.

3.1.15 *JavaScript* објекти, особине и методе

У стварном животу постоје објекти (*objects*), који имају своје особине (*properties*) и могу да врше неке радње (извршавају функције, имају придружене методе, *methods*). Објектно програмирање би требало што је могуће верније да буде прилагођено реалним системима који се састоје од великог броја објеката, где се по потреби могу појављивати нове особине и могућности или губити нека својства код објеката насталих од постојећих. Тако на пример ако посматрамо два објекта која треба моделовати у програмском коду, нпр. човек и ауто, тада они имају особину боје, код човека боју очију а код аута боју каросерије. Ако треба декларисати променљиву у коду која означава вредност боје, тада је потребно да се зна и на који објекат се односи боја. Најједноставнији начин да се променљива која садржи информацију о боји придружи објекту, тако да се лако идентификује ком објекту припада променљива са особином објекта.

Као што се код генерисања назива променљивих користи уобичајени шаблон да се назив састоји од већег броја речи које су спојене као једна реч, а разликује се почетак сваке нове речи тиме што је написана великим словом, тако се и за објекте мора дефинисати прецизна структура означавања како при кодовању не би дошло до двозначности, а самим тим и до грешке, при интерпретацији.

JavaScript објекти су илустровани на примеру са слике 3.1.33. Нека је као пример објекта ауто са слике 3.1.33 визуелно представљен сликом аутомобила.



Слика 3.1.33 Објекти, особине и методе.

Овај објекат има више особина и назив особине је раздвојен тачком. Назив објекта је ауто, а назив особине је боја.

Назив променљиве по раније описаним правилима би био `autoBoja`, а вредност која се додељује би била стринг `"crvena"`:

```
var autoBoja = "crvena";
```

У ситуацији када постоји мали број променљивих, овакав запис у коду би био сасвим прикладан. Проблем настаје када се значајно повећа број променљивих, па би програмер увек морао да има на уму која променљива припада ком објекту. За одржавање софтвера ово представља још већи проблем, зато што неко ко тестира и контролише исправност кода мора да има исте предпоставке као и програмер који је написао оригиналан код. И сами програмери после неког времена од писања програма, забораве шта су хтели.

JavaScript променљива је контејнер који садржи вредност, а контејнер може да садржи већи број вредности, структурираних тако да се лако проналазе све информације из контејнера.

Зато је направљен посебан тип података који се зове објекти, са циљем да се лакше проналазе вредности које припадају једном објекту.

3.1.15.1 *JavaScript* објекти

За разлику од обичних променљивих, објекти могу да садрже већи број вредности, при чему вредности могу да буду и различите по типовима. На пример:

```
var auto = {ime:"Fiat", tezina:850, boja:"crvena"};
```

Све особине се налазе између витичастих заграда, {}, а све вредности су раздвојене двотачком : од особине. Парови вредности особина:вредност су раздвојени зарезима и називају се особинама објекта. У овом примеру објекат је променљива којој је додељене већи број вредности. Вредности могу да буду стринг, "Fiat" или "crvena", али и број 850. Уколико се ради о особини за коју важи исто својство, на пример да је изражено у истим мерним јединицама, тада се особини може доделити само број 850, а не цео стринг "850 kg". Оператори са стринговима могу бити компликовани ако су потребна аритметичка израчунавање, па је добро да се користе бројеви уместо стрингова за доделу вредности променљивој. Особине објеката се пишу као парови назив:вредност, раздвојени зарезима:

```
var osoba = {ime:"Miro", prezime:"Lu", godiste:50, ociBoja:"zelena"};
```

Објекат који има назив osoba у овом примеру има 4 особине: ime, prezime, godiste, ociBoja. Вредности су у овом примеру типа стринг и број.

Оператор typeof може да се користи да се одреди тип *JavaScript* промењиве. тако што враћа тип промењиве или израз. У *JavaScript*-у, тип података који је null за typeof враћа резултат да је тип објекат.

Ако је декларисан објекат, па му је додељена вредност null, тада он и даље остаје објекат.

```
var osoba = {ime:"Miro", prezime:"Lu", godiste:50, ociBoja:"zelena"};
osoba = null;    // vrednost je null, ali je tip objekat
```

Разлика између undefined и null може се разумети ако се погледа пример из кога произилази да је за null вредност недефинисана, али да је тип податка објекат:

```
typeof null          // object
null === undefined   // false
null == undefined    // true
```

Сложени типови података су они који за typeof оператор враћају тип object. На пример:

```
typeof {ime:'Miro', god:34}  // vraca rezultat "object"
typeof [1,2,3,4]             // za "array" vraca rezultat "object"
typeof null                  // vraca rezultat "object"
```

Оператор typeof враћа "object" за низ зато што се низ третира као објекат.

При додели вредности објекту, бланко карактери и прелазак у нови ред немају значај:

```
var osoba = {
  ime:"Miro",
  prezime:"Lu",
  godiste:50,
  ociBoja:"zelena"
};
```

3.1.15.1 Приступ особинама *JavaScript* објеката

JavaScript особинама објеката може да се приступи на два начина, тачком после назива објекта са особином објекта у наставку

```
imeObjekta.osobinaObjekta
```

или назив објекта после кога је у угластим заградама особина чију вредност треба прочитати

```
imeObjekta["osobinaObjekta"]
```

Пример је илустрован на слици 3.1.34.

Слика 3.1.34 Приступ особинама *JavaScript* објеката.

Први део стринга који приказује прегледач `"Miro"` добијен је приступом са тачком између објекта `osoba` и једне од особина објекта, `ime`. Други део стринга `"Lu"`је добијен коришћењем угластих заграда и знака наводника између којих је друга особина објекта `prezime`. Између ова два стринга уметнут је један бланко знак `" "`.

3.1.15.2 Приступ методама *JavaScript* објеката

JavaScript методе објеката су акције које објекти могу да изврше. Дефиниције функција се чувају као особине објеката. На пример:

```
var osoba = {
  ime : "Miro",
  prezime : "Lu",
  godiste : 50,
  imePrezime : function() {
    return this.ime + " " + this.prezime;
  }
};
```

Особине објекта `osoba` су `ime`, `prezime`, `godiste` и `imePrezime`. Прве три особине имају додељену вредност, типа стринг или број. Последња особина користи кључну реч `function()` и има додељену функцију тиме што користи отворену и затворену обичну заграду која симболички означава да ће бити искоришћени неки параметри да би се извршила нека израчунавања. У витичастим заградама се налазе наредбе које треба да се изврше када се позове функција. Када функција израчуна то што је њен задатак, вредност се враћа тамо где је позвана функција, а у овом случају то је вредност која ће бити придружена називу `imePrezime`. Користи се кључна реч `this`, да се означи да називи који се користе после `this` са тачком, `this.`, су особине објекта у оквиру кога се дефинише функција. Уместо да се користи особина објекта `osoba.ime`, користи се `this.ime`, уместо да се користи особина објекта `osoba.prezime`, користи се `this.prezime`. Наредба која се

налази после речи `return`, извршава конкатенацију стрингова имена, бланко знака и презимена и генерише нови стринг. Овако добијен стринг ће се појавити када буде позвана особина `imePrezime`.

Када се додељује нека вредност промењивој, или се одређује особина објекта, не треба поново вршити доделу у програмском коду. Ако би се на више места у коду уносила иста вредност, тада би могло да се догоди да програмер грешком уради измену на једном месту и да заборави да уради измену и на свим осталим местима у коду. Другим речима додела вредности треба да буде само на једном месту у коду. Кључна реч `this` се користи уместо назива објекта, да би се само на једном месту генерисао назив промењиве, у овом случају `osoba`. Функција се користи у оквиру објекта за израчунавања при чему се користе једном придружене особине објекта. Ако би се променила вредност стринга за неку особину, тада би се и сва израчунавања извршавала са промењеном вредношћу особине.

Пример је илустрован на слици 3.1.35. На слици десно је приказ прегледача резултата методе.

Слика 3.1.35 Приступ особинама *JavaScript* објеката.

У случају да се метода позове без обичних заграда, на пример у линији 16 кода са слике 3.1.35 са `osoba.imePrezime;` уместо са `osoba.imePrezime()`, у прегледачу ће бити приказана дефиниција методе на исти начин као што је то било у примеру *JavaScript* функције. Тиме се потврђује да се метода објекта третира као особина објекта ако се позива без заграда `()`.

3.1.15.3 Декларисање *JavaScript* објеката

JavaScript објекти се декларишу коришћењем кључне речи `new`. Ову кључну реч не треба користити за декларисање бројева, стрингова или логичких промењивих зато што то успорава извршења кода. На пример, не препоручује се следеће декларисање промењивих:

```
var x = new String();        // Deklarise x kao string objekat
var y = new Number();        // Deklarise y kao brojni objekat
var z = new Boolean();       // Deklarise z kao logicki objekat
```

3.1.16 *JavaScript* догађаји

HTML догађај (*events*) је "нешто (*things*)" што се догађа *HTML* елементима. Када се користи *JavaScript* у *HTML* страници, *JavaScript* може да реагује на овакав догађај. *HTML* догађај је нешто на шта ће прегледач или корисник урадити, на пример:

- *HTML* веб страна је завршила са учитавањем.
- *HTML* улазно поље је промењено.
- *HTML* дугме је кликнуто.

Када се догоди *HTML* догађај је, може да се изврше неке наредбе. *JavaScript* омогућава да се изврши део програмског кода када се детектује неки догађај. Када се *HTML* елементима дода атрибут као такозвани *event handler*, прегледач проналази *JavaScript* код и извршава наредбе на месту где је постављен идентификатор. Могу се користити једноструки и двоструки наводници; на пример атрибути за *HTML* елемент могу да буду:

```
<element event='JavaScript kod'>
<element event="JavaScript kod">
```

У овом примеру уместо *element* може да буде неки *HTML* таг:

Догађај (*event*)	Опис
onchange	*HTML* елемент је промењен
onclick	Корисник је кликнуо на *HTML* елемент
onmouseover	Корисник је померио миша изнад *HTML* елемента
onmouseout	Корисник је померио миша изван *HTML* елемента
onkeydown	Корисник је притиснуо типку на тастатури
onload	Прегледач је завршио са учитавањем веб стране

У следећем примеру је показано када је
за *element* изабран таг `button`, са атрибутом у коме је
за *event* изабран догађај `onclick`,
за *"JavaScript kod"* изабран `"document.getElementById('demo').innerHTML = Date()";`,
једноструки наводници се користе за идентификатор `'demo'`;
текст који се исписује на дугмету је `Datum je`;
прегледач који извршава *JavaScript* код на месту позива по идентификатору исписује вредност коју враћа метода `Date()`.

```
<button onclick="document.getElementById('demo').innerHTML = Date()">Datum je</button>
```

Ако се мења садржај самог елемента у оквиру кога се реагује на догађај, на пример да се мења текст који се исписује на дугмету, тада се користи кључна реч `this`, на пример:

```
<button onclick="this.innerHTML = Date()">Datum je</button>
```

У овом примеру иницијално на дугмету пише `Datum je`, а након клика на дугмету, уместо овог текста се појављује резултат методе, на пример у време писања овог поглавља

```
Fri Jul 24 2020 09:39:13 GMT+0200 (Central European Summer Time)
```

Да би код био једноставнији, може да се користи и следећа синтакса:

```
<button onclick="displayDate()">The time is?</button>
```

На овом месту биће поновљено шта све може да се уради са *JavaScript* –ом у ситуацијама када треба да се реагује на неки догађај, на пример:

- Да се уради сваки пут када се учита веб страна.
- Да се уради сваки пут када се затвори веб страна.
- Да се уради када се кликне на дугме.
- Да се провери садржај када корисник унесе податке.

За то што треба *JavaScript* да уради када се догоди неки од наведених догађаја, *JavaScript* користи различите методе, на пример:

- Атрибут *HTML* догађаја може директно да изврши *JavaScript* код.
- Атрибут *HTML* догађаја може да позове *JavaScript* функцију.
- За управљање догађајем, може се доделити сопствена функција *HTML* елементу
- Може се спречити догађај да буде послат или да се њиме управља.

3.1.17 *JavaScript* стрингови

JavaScript стрингови се користе за складиштење и разне операције са текстовима. Стрингови се уносе између знакова навода и могу да имају велики број карактера али и да буде празан стринг, односно између стрингова нема ни једног карактера, на пример:

```
var string1 = "Miro Lu";   // koriscenje dvostrukih navodnika
var string2 = 'Miro Lu';   // koriscenje jednostrukih navodnika
var string3 = " ";         // string sa blanko karakterom
var string4 = "";          // prazan string
var string5 = '';          // prazan string
```

Ако стринг треба да садржи један тип наводника, на пример једноструке наводнике, тада се користи други тип наводника између којих је цео стринг. Први наводници (без обзира да ли су једноструки или двоструки) одређују почетак стринга, а када се понови исти тип наводника, он означава крај стринга. На овај начин се први наведени наводник понаша као да је отворена заграда и када се понови појави исти тип наводника, тада има улогу затворене заграде. Између првог и другог наводника исте врсте који се понашају као отворена и затворена заграда, може се користити други тип наводника као да је обичан карактер. На пример:

```
var string6 = " Da l' sam u pravu?";
var string7 = "jednostruki navodi 'izmedju' dvostrukih";
var string8 = 'dvostruki navodi "izmedju" jednostrukih';
```

У прегледачу ће бити приказана ова три стринга на следећи начин;

```
Da l' sam u pravu?
jednostruki navodi 'izmedju' dvostrukih
dvostruki navodi "izmedju" jednostrukih
```

То значи да ако је потребно у прегледачу приказати двоструке наводнике, као за стринг `string8`, тада се цео стринг записује између једноструких наводника. Ако је потребно у прегледачу приказати једноструке наводнике, као за стрингове `string6` и `string7`, тада се цео стринг записује између двоструких наводника.

Иако се стрингови декларишу као тип података стринг, могу се користи кључне речи у формату који се користи као особина објекта са тачком, на пример:

```
var string7 = "jednostruki navodi 'izmedju' dvostrukih";
var duzinaString7 = string7.length;
```

Након декларисања променљиве типа стринг `string7`, и доделе вредности променљивој, додавањем тачке и кључне речи `length`, односно `string7.length`, добија се број који даје вредност броја карактера који је садржан у стрингу. За наведени пример променљивој `duzinaString7` биће додељена вредност 39. За празан стринг (на пример за `string4` и `string5`), за дужину стринга се добија вредност 0.

3.1.17.1 *Escape* карактери у *JavaScript* стринговима

Обзиром да се наводници користе да означе да је тип података стринг, може доћи до погрешне интерпретације код прегледача. На пример при декларисању променљиве типа стринг могу се употребити исти типови наводника:

```
var string9 = "jednostruki navodi "izmedju" dvostrukih";
```

Ако се користи едитор који користи боје за *JavaScript* кодовање, текст који је између других двоструких наводника биће у другачијој боји; једна боја се користи за `"jednostruki navodi "` и `" dvostrukih"`, док је другачије боје за текст `izmedju`. Да је уместо речи `izmedju` био знак `+`, тада би била извршена конкатенација два стринга. Ако би уместо речи `izmedju` био текст који није *JavaScript* оператор или *JavaScript* кључна реч, тада прегледач највероватније не би урадио то што је програмер хтео, већ би та линија кода била игнорисана и прегледач не би ништа приказао.

Да би се избегли овакви проблеми, треба користи такозване ескејп карактере, *escape character*, У овом случају се користи обрнута коса црта (коса црта која је укошена на другу страну у односу на оператор подељено), на енглеском *backslash escape character*.

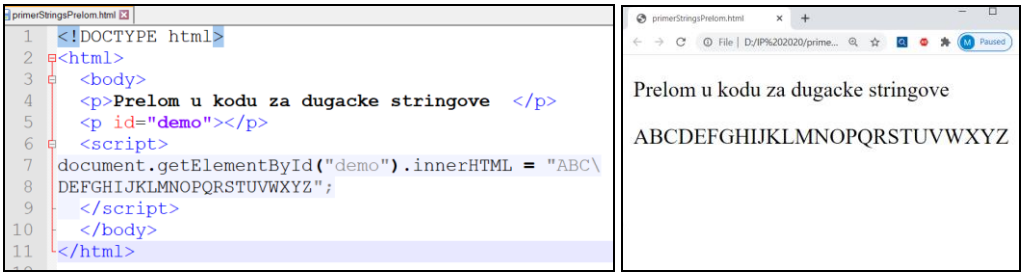
У следећој табели су приказани *JavaScript* кодови који као резултат приказују у прегледачу одговарајући резултат и опис значења у *JavaScript*-у.

Код	**Карактер обрнута коса црта, *backslash escape character***	
	Резултат	**Опис**
`\'`	'	Једноструки наводници, *single quote*
`\"`	"	Двоструки наводници, *double quote*
`\\`	\	Обрнута коса црта, *backslash*

Постоје још неки ескејп карактери који су првобитно нили намењени писаћим и факс машинама, а за приказ у прегледачу немају никакав значај, односно прегледач ће их игнорисати у *HTML* фајлу.

Код	**Ескејп карактери које не користи прегледач**	
	Резултат	**Опис**
`\b`	`Backspace`	Померај тренутне позиције за једно место у лево
`\f`	`Form Feed`	Прелазак на следећу страну
`\n`	`New Line`	Крај линије и прелазак у нову линију
`\r`	`Carriage Return`	Ресетовање позиције на почетак линије
`\t`	`Horizontal Tabulator`	Хоризонтални табулатор
`\v`	`Vertical Tabulator`	Вертикални табулатор

У претходном делу, у више примера је показано како се дуге линије кода могу преломити у више линије а да то не произведе грешку при интерпретацији у прегледачу. Показано је. такође, да стрингови не могу да се приказују у више линија програмског кода једноставним преласком у нови ред, зато што би наредна линија била третирана као да је нова наредба. Да стринг могао исправно да се прикаже у више линија кода потребно је унети обрнуту косу црту на месту где стринг може да се преломи и прикаже у новом реду, као што је показано у следећем примеру на слици 3.1.36.



Слика 3.1.36 Прелом линије у коду за *JavaScript* стринг.

Линија кода број 7 у којој је требао да се налази стринг `"ABCDEFGHIJKLMNOPQRSTUVWXYZ"` била би превише дугачка јер испред стринга је део наредбе која треба да прикаже стринг у прегледачу. Убачена је обрнута коса црта после слова `C` а испред слова `D` и остатак стринга је пренет у линију кода број 8.

```
"ABC\
DEFGHIJKLMNOPQRSTUVWXYZ"
```

Прегледач приказује цео стринг као да је написан стринг у линији кода 7 без преласка у линију кода 8, а не приказује косу црту која није део стринга већ се користи као ескејп карактер.

Неки прегледачи не дозвољавају коришћење обрнуте косе црте, па је пожељно да се користе други начини. Ако се не користи прелом линије унутар стринга, најбоље је да се користи прелом после *JavaScript* оператора који је изван *JavaScript* стринга. На пример за слику 3.1.36, програмски код би могао да буде:

```
document.getElementById("demo").innerHTML = "ABC" +
"DEFGHIJKLMNOPQRSTUVWXYZ";
```

У овом случају *JavaScript* оператор + је искоришћен да уради конкатенацију два стринга, `"ABC"` и `"DEFGHIJKLMNOPQRSTUVWXYZ"`, а дугачки стринг је написан у две линије *JavaScript* кода. Добија се идентичан резултат као на слици 3.1.36 десно.

Треба бити опрезан са коришћењем обрнуте косе црте \ зато што се она користи у стрингу који прелама стринг на два дела. Изван стринга никако не треба да се користи обрнута коса црта за прелазак линије са *JavaScript* наредбом, јер то није намена ескејп карактера.

3.1.17.2 Декларисање *JavaScript* стринга као објекат

JavaScript стрингови се уобичајено декларишу као променљиве типа стринг што се означава знаковима навода. То су примитивне вредности за разлику од сложених типова као што су објекти, на пример:.

```
var string1 = "Miro Lu";   // koriscenje dvostrukih navodnika
var string2 = 'Miro Lu';   // koriscenje jednostrukih navodnika
```

JavaScript стрингови могу да се декларишу као објекти и тада се користи кључна реч `new`:

```
var imeOsobe = new String("Miro");
```

Када се користи кључна реч `typeof`, стринг `string1` враћа `string`, а стринг `imeOsobe` враћа `object`, а то значи да `string1` и `imeOsobe` нису исти тип података иако је садржај стринга идентичан. Операције са примитивним вредностима типа стринг се брже извршавају него са сложеним типовима типа објекат, чак и ако је исти садржај стринга.

Треба избегавати декларисање стрингова као објекат, јер ће то успорити извршавање програмског кода, а и резултати извршавања могу бити другачији од тога шта програмер очекује због компликованијег кода. На пример ако се тестирају вредности променљивих коришћењем оператора `==`

```
(string1 == imeOsobe)
```

добиће се да су ова два стринга иста (оператор `==` даје вредност `true`).

Ако се тестирају вредности променљивих коришћењем оператора `===`

```
(string1 === imeOsobe)
```

добиће се да ова два стринга нису иста (оператор `===` даје вредност `false`) зато што је један типа стринг а други типа објекат.

Посматрајмо пример:

```
var imeOsobe1 = new String("Miro");
var imeOsobe2 = new String("Miro");
```

Тестирање вредности коришћењем оператора `==` и `===`

```
(imeOsobe1 == imeOsobe2)
(imeOsobe1 === imeOsobe2)
```

добиће се да нису исти (оператори `==` и `===` дају вредност `false`), иако је идентичан садржај стрингова и оба су истог типа, објекат. Да се не би дешавале овакве грешке, најбоље је да се не декларишу стрингови као објекти.

3.1.18 *JavaScript* стринг особине и методе

JavaScript стрингови декларисане као примитивне вредности типа стринг не могу да имају особине и методе, зато што нису типа објекат. Међутим *JavaScript* омогућава да се особине и методе могу користити и за примитивне вредности зато што *JavaScript* третира примитивне вредности као објекте када извршава особине и методе. Ово значајно олакшава рад са стринговима. Један такав пример је приказан у претходном поглављу:

```
var string7 = "jednostruki navodi 'izmedju' dvostrukih";
var duzinaString7 = string7.length;
```

Након декларисања променљиве типа стринг и доделе вредности променљивој, коришћењем кључне речи `length` добија се број који даје вредност броја карактера који је садржан у стрингу.

Метода `indexOf()` враћа позицију првог појављивања текста који је наведен између обичних заграда. Број који се добија назива се и индекс појављивања. На пример,

```
var string7 = "jednostruki navodi 'izmedju' dvostrukih";
var pozicija7 = string7. indexOf ("struki");
```

Део стринга `"jednostruki"` садржи текст `"struki"`, а прво слово `"s"` се појављује на позицији 5 у стрингу `string7`. Зато ће вредност доделе променљивој `pozicija7` бити број 5. Важно је уочити да је најмања вредност индекса 0 ако се текст поклапа са почетком стринга, па тако индекс за слово `j` у тексту `"jednostruki..."` одговара позицији 0, слову `e` индекс 1, слову `d` индекс 2, слову `n` индекс 3, слову `o` индекс 4, и коначни слову `s` индекс 5. У случају да стринг не садржи тражени текст, индекс добија вредност -1.

Метода `lastIndexOf` враћа позицију последњег појављивања текста који је наведен између обичних заграда. На пример,

```
var string7 = "jednostruki navodi 'izmedju' dvostrukih";
var pozicija7last = string7.lastIndexOf("struki");
```

Део стринга `"dvostrukih"`, садржи два пута текст `"struki"`, а код последњег појављивања траженог текста прво слово `"s"` из текста `"struki"` се појављује на позицији 32 у стрингу `string7`. Зато ће вредност доделе променљивој `pozicija7last` бити број 32 као последњи почетак траженог текста. Као и у случају методе `indexOf` тако и у случају методе `lastIndexOf`, индекс добија вредност -1 када стринг не садржи тражени текст.

Обе методе, `indexOf` и `lastIndexOf`, могу да имају два параметра, при чему други параметар одговара позицији од које почиње да претражује тражени текст.

```
var string7 = "jednostruki navodi 'izmedju' dvostrukih";
var pozicija7 = string7. indexOf ("struki",15);
```

Део стринга `"jednostruki"` садржи два пута текст `"struki"`, при чему је прво појављивање на позицији 5 а друго на позицији 32. Ако је задат други параметар 15, тада претрага почиње од позиције 15 и први пут проналази текст на позицији 32.

За методу `lastIndexOf`, која има и други параметар, на пример 15, враћа позицију последњег појављивања текста који је наведен између обичних заграда полазећи од позиције 15 и претражујући ка почетку. На пример,

```
var string7 = "jednostruki navodi 'izmedju' dvostrukih";
var pozicija7last = string7.lastIndexOf("struki",15);
```

Сада је последњи пут пронађен текст на позицији 5. Као и у случају методе са једним параметром, индекс добија вредност -1 када део стринга не садржи тражени текст при претраживању од вредности другог аргумента ко краја за `indexOf` или до почетка за `lastIndexOf`.

3.1.18.1 Проналажење стринга у *JavaScript* стринговима

Метода `search()` претражује стринг за задату вредност и враћа позицију појављивања за вредност која је наведен између обичних заграда. На пример,

```
var string7 = "jednostruki navodi 'izmedju' dvostrukih";
var pozicija7 = string7.search("struki");
```

Добија се исти резултат као када је коришћена метода `indexOf`, а то је број 5. Индекс је позиција када се у стрингу `"jednostruki..."` појављује први пут прво слово `"s"` од стринга `"struki"`, а је број 5.

Из овог примера би се могло закључити да су ове две методе, `search` и `indexOf`, идентичне зато што се за претрагу по истом стрингу `string7`, као параметар користи исти аргумент `"struki"`, и добија исти број 5 као резултат.

Међутим, ова два метода нису иста и постоје разлике:

- Метода `search()` не може да има два аргумента, односно не може да се зада индекс за почетак претраживања.
- Метода `indexOf()` не може да прихвати сложеније вредности за претраживање као што су регуларни изрази.

За издвајање делова стринга користе се три методе:

- `slice(`*`start, end`*`)`
- `substring(`*`start, end`*`)`
- `substr(`*`start, length`*`)`

Метода `slice()` издваја део стринга и враћа издвојени део новом стрингу. Метода има два параметра, почетну и последњу позицију у стрингу из кога се издваја део као нови стринг. Последња позиција је за 1 мања од другог атрибута. У следећем примеру се издваја део стринга од позиције 5 до 10 где је 10=(11-1):

```
var string7 = "jednostruki navodi 'izmedju' dvostrukih";
var izdvojeni7p = string7.slice(5, 11);
```

променљивој `izdvojeni7p` се додељује издвојени део стринга `"struki"`. Напомена, у прегледачу се приказује само текст `struki`. Зато што знаци навода у програмском коду служе да означе да се ради о примитивној вредности типа стринг, а наводници нису део издвојене вредности.

Уобичајена грешка која се дешава јесте да програмер придружује број 1 првој позицији стринга; *JavaScript* броји од нуле, па за прву позицију стринга користи број 0.

Ако аргументи имају негативне вредности, тада се броји од краја стринга ка почетку стринга, на пример од позиције -34 до позиције -28:

```
var string7 = "jednostruki navodi 'izmedju' dvostrukih";
var izdvojeni7m = string7.slice(-34, -28);
```

Дужина стринга је 39. Први карактер у стрингу `string7` рачунајући са десна у лево је на позицији -39, а последњи је на позицији -1. Када се рачуна позиција са лева у десно, први карактер је на позицији 0, а последњи је на позицији 38. За анализиране примере

```
var string7 = "jednostruki navodi 'izmedju' dvostrukih";
var izdvojeni7p = string7.slice(0, 3);
var izdvojeni7m = string7.slice(-39, -36);
```

Код рачунања са позитивним позицијама, прва три карактера се добијају са `slice(0, 3)`, зато што првој позицији стринга `string7` одговара индекс 0 а трећој позицији одговара 2 (добијено умањењем другог аргумента за 1, 2=3-1). Код рачунања са негативним индексима, прва три карактера се добијају са `slice(-39, -36)`, зато што првој позицији одговара аргумент-39. а трећој позицији одговара -36. У оба случаја се добија вредност прва три карактера у стрингу `string7`, а то је стринг `"jed"`.

Ако се изостави други параметар, подразумева се да је за први позитиван аргумент други аргумент последњи у стрингу; подразумева се да је за први негативан аргумент други аргумент први у стрингу. За анализирани пример добија се цео стринг ако се користи `slice(0)`, као и за `slice(-39)`.

Треба бити опрезан са коришћењем негативних позиција, зато што га не подржавају сви прегледачи.

Метода `substring()` је слична методи `slice()`, осим што не подржава негативне индексе. Ако се изостави други параметар, подразумева се да је други аргумент последњи у стрингу. На пример

```
var string7 = "jednostruki navodi 'izmedju' dvostrukih";
var izdvojeni7 = string7.substring(0, 3);
var izdvojeni7ceo = string7.slice(0);
```

Издвојени стринг `izdvojeni7` ће садржати прва три карактера стринга `string7`. Издвојени стринг `izdvojeni7ceo` ће садржати цео стринга `string7`.

Метода `substr()` се разликује од методе `substring()` по томе што се другим параметром одређује дужина стринга који се издваја. Ако се изостави други параметар, подразумева се да ће други аргумент одговарати дужини до краја стринга. На пример

```
var string7 = "jednostruki navodi 'izmedju' dvostrukih";
var izdvojeni7d6 = string7.substr(5, 6);
var izdvojeni7doKraja = string7.substr(5);
```

Издвојени стринг `izdvojeni7d6` ће садржати стринг `"struki"`. Издвојени стринг `izdvojeni7doKraja` ће садржати остатак стринга после позиције 5 у стрингу `string7`, а то је `"struki navodi 'izmedju' dvostrukih"`.

Метода `substr()` за разлику од методе `substring()` може да има негативан први индекс. а други параметар одређује дужина стринга који се издваја. Ако се изостави други параметар, подразумева се да ће други аргумент одговарати дужини до краја стринга у десно (а не у лево до почетка стринга). На пример

```
var string7 = "jednostruki navodi 'izmedju' dvostrukih";
var izdvojeni7d6 = string7.substr(-34, 6);
var izdvojeni7doKraja = string7.substr((-34);
```

У овом случају, за `substr(-34, 6)` и `substr(-34)`, се добијају исти резултати као да су коришћени `substr(5, 6)` и `substr(5)`, зато што за стринг дужине 39 позиција са

позитивним аргументом 5 показује на исту позицију као и аргумент-39. Ако је други параметар изостављен, тада се подразумева да је дужина до краја стринга, а то је 34=39-5.

3.1.18.2 Замена садржаја стринга

Метода `replace()` замењује специфицирану вредност са другом вредношћу у стрингу. Ова метода не мења стринг за који је позвана, већ додељује новој променљивој стринг у коме је извршена промена, По правилу, подразумева се да ће метода извршити замену само за прво поклапање са вредношћу која се мења. На пример:

```
var str = "jednostruki navodi 'izmedju' dvostrukih";
var txt = str.replace("struki", "NestoNovo");
```

На слици 3.1.37 приказан је пример замене дела једног стринга другим текстом, а затим стринг са промењеним садржајем приказан у прегледачу.

Слика 3.1.37 Пример замене садржаја стринга коришћењем функције.

На слици горе лево је код. У линији кода број 4 написано је у параграфу шта ради овај пример. Прегледач приказује на слици горе десно изглед када се отвори фајл у прегледачу. Линија број 5 садржи приказ дугмета са описом у њему шта се дешава када се кликне. Атрибут дугмета, `onclick="myFunction()"`, дефинише шта се дешава када се дугме кликне, а то је позив функције `"myFunction()"`. Линија број 6 има идентификатор, `id="demo"`, као атрибут параграфа. Параграф приказује неки текст.

Када се први пут кликне дугме, он извршава функцију која је дефинисана између скрипт тагова. Локалној променљивој `str` се додељује садржај параграфа који је у линији кода број 6, `"jednostruki navodi 'izmedju' dvostrukih"`. У линији кода број 10 се врши замена вредности у `str`, тако што се тражи са лева у десно прво појављивање вредности `"struki"`. Када се пронађе овај део стринга, он се замењује са `"NestoNovo"`. Сада се у параграфу исписује нови стринг `"jednoNestoNovo navodi 'izmedju' dvostrukih"`, уместо `"jednostruki navodi 'izmedju' dvostrukih"`, као што је приказано у прегледачу на слици доле лево. Када се други пут кликне дугме, променљивој `str` се додељује нови

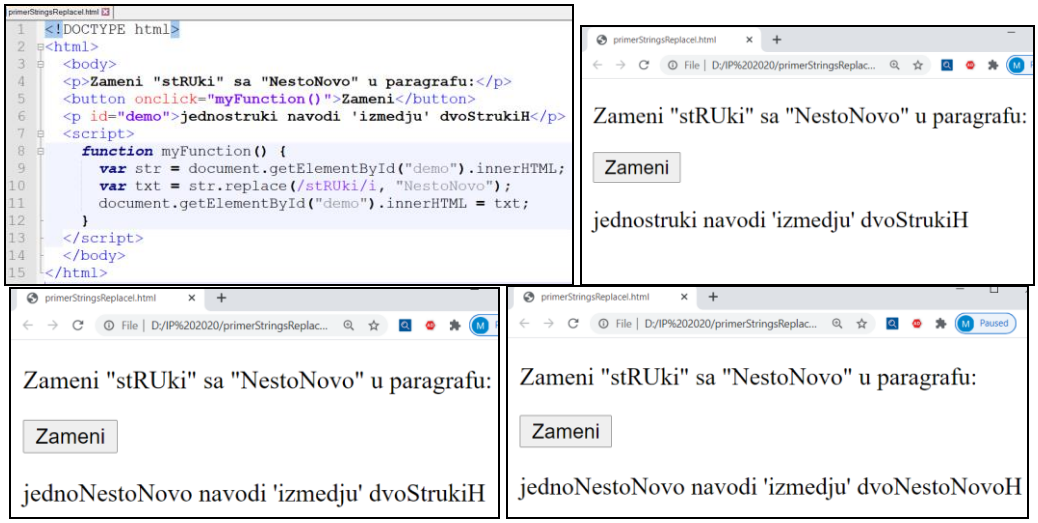
стринг `"jednoNestoNovo navodi 'izmedju' dvostrukih"`, у новом стрингу се тражи шта треба да се замени, а то је први пут у последњој речи `dvostrukih`. Врши се замена дела текста и добија се нови стринг `"jednoNestoNovo navodi 'izmedju' dvoNestoNovoh"`. Сада се мења садржај параграфа новим стрингом који приказује у прегледачу на слици доле десно.

Функција `myFunction()`, има две локалне променљиве. Прва локална променљива је `str` којој се додељује тренутни садржај параграфа који има као атрибут идентификатор `id="demo"`. Друга локална променљива је `txt` којој се у функцији додељује вредност после замене дела садржаја `str` новом вредношћу. Када се кликне дугме, функција замењује текст у параграфу новим стрингом који је у локалној променљивој `txt`. Када се други пут кликне на дугме, правој локалној променљивој `str` се додељује тренутни садржај параграфа, а то је онај који је био у `txt`. Функција проналази да постоји тражена вредност у последњој речи, и мења садржај параграфа. Када се се још једном кликне на дугме, локалној променљивој `str` се додељује тренутна вредност а у њој више не постоји тражена вредност. Локална променљива `txt` остаје иста као и `str`. Колико год пута да се кликне дугме, стринг у параграфу остаје непромењен зато што не садржи вредност коју треба променити.

Метода `replace()` је *case sensitive*, односно велико и мало слово су различите вредности. То значи да ако реч која се тражи има макар једно слово које је другачије по величини слова у стрингу у коме треба извршити замену, замена неће бити извршена.

Да би се извршила замена у стрингу, потребно је да се користе регуларни изрази са флегом `/i` (*insensitive*) , на пример:

```
var str = "jednostruki navodi 'izmedju' dvoStrukiH";
var txt = str.replace(/stRUki/i, "NestoNovo");
```



Слика 3.1.38 Пример замене садржаја стринга коришћењем израза.

У коду са слике 3.1.38 горе лево је коришћен израз `stRUki`, уместо стринга `"struki"` као у примеру са слике 3.1.37. Израз се налази између косих црта (као да је оператор дељења) при чему после друге косе црте стоји мало слово и, `/i`. Ово значи да ће сви стрингови који садрже слова као у стрингу `struki`, бити замењени новим стрингом, `"NestoNovo"`, без обзира да ли је неко слово мало или велико. Након учитавања *html* фајла, добија се приказ у прегледачу као на слици 3.1.38 десно, после првог клика на дугме се добија слика доле лево, а после другог клика се добија слика доле десно. Сваки наредни клик не доноси нове замене, зато што текст у пасусу са идентификатором не садржи израз који треба променити. Све остало функционише као што је објашњено у примеру са слике 3.1.37.

Да би се урадиле замене свих израза одједном, без потребе да се проналази сваки наредни израз, треба користити флег `/g`, који означава глобално поклапање *global match*. На пример:

```
var str = "jednostruki navodi 'izmedju' dvostrukih";
var txt = str.replace(/struki/ g, "NestoNovo");
```

За конверзију малих слова у велика користи се метода `toUpperCase()`. За конверзију великих слова у мала користи се метода `toLowerCase()`, на пример:

```
var text1 = "Miro Lu";            // String
var text2 = text1.toUpperCase();  // rezultat je MIRO LU
var text3 = text1.toLowerCase();  // rezultat je miro lu
```

За конкатенацију стрингова се користи метода `concat()`:

```
var text1 = "Miro";
var text2 = "Lu";
var text3 = text1.concat(" ", text2); // rezultat je string Miro Lu
```

Уместо оператора `+`, исти резултат се добија методом `concat()`, на пример:

```
var text4 = "Miro" + " " + "Lu";      // rezultat je string Miro Lu
var text5 = "Miro".concat(" ", "Lu"); // rezultat je string Miro Lu
```

Све стринг методе враћају нови стринг, што значи да не мењају оригинални стринг на коме се извршава метода. Формално се каже да је стринг непромењив (*immutable*), што значи да стринг не може да се промени, већ може да се замени другим стрингом.

Метода `trim()` уклања бланко карактере у стрингу са леве и десне стране, на пример:

```
var str = "    Miro Lu    ";
alert(str.trim());
```

```
<!DOCTYPE html>
<html>
  <body>
  <p>trim() metoda brise blanko levo i desno:</p>
  <p id="demo"></p>
  <script>
    var text1 = "    Miro Lu    ";
    var text2 = text1.trim();
    var text3 = "||"+text1+"||"+"<br>||"+text2+"||";
  document.getElementById("demo").innerHTML = text3;
  </script>
  </body>
</html>
```

Слика 3.1.39 Пример методе `trim()`.

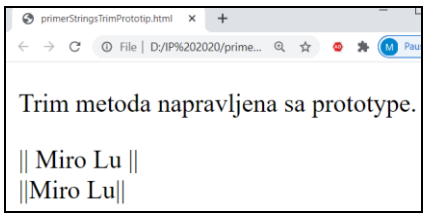
На слици 3.1.39 је показано како метода `trim()` уклања бланко карактере из стринга. Иако стринг `text1` има велики број бланко карактера са леве и десне стране текста у стрингу, у прегледачу се види као да постоји само по један бланко знак лево и један бланко знак десно од текста у стрингу, зато што *HTML* игнорише велики број бланко знакова и празне линије и приказује само један бланко знак. Да би било јасније шта ради ова метода, приказују се по две вертикалне црте испред и после стринга. Након позивања методе `trim()`, стринг више не садржи ни један бланко знак, па је текст приказан до вертикалних црта.

Метода `trim()` не функционише у свим врстама прегледача, на приме у *Internet Explorer* 8.

Ако је потребно да се дода функција која реализује неку методу, то може да се уради са `String.prototype`, на пример:

```
if (!String.prototype.trim) {
  String.prototype.trim = function () {
    return this.replace(/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g, '');
  };
}
```

```
<!DOCTYPE html>
<html>
  <body>
  <p>Trim metoda napravljena sa prototype.</p>
  <p id="demo"></p>
  <script>
    if (!String.prototype.trim) {
      String.prototype.trim = function () {
return this.replace(/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g, '');
      };
    }
    var text1 = "    Miro Lu    ";
    var text2 = text1.trim();
    var text3 = "||"+text1+"||"+"<br>||"+text2+"||";
    document.getElementById("demo").innerHTML = text3;
  </script>
  </body>
</html>
```



Слика 3.1.40 Пример методе `trim()` направљене са `prototype`.

На слици 3.1.40 показано је како може да се направи сопствена метода за уклањање бланко знакова са леве и десне стране коришћењем `prototype`. Десни део слике показује да се добија исти приказ као и коришћењем методе `trim`. Хексадецимална ознака за

бланко ознаку је `A0`, као *html* ознака се користи ` ` а што се добија од великих слова у енглеској терминологији од ` `. Метода `replace` претражује са леве и десне стране све бланко карактере стринга и замењује их са празним стрингом, што је исто као да брише бланко карактере из стринга. .

3.1.18.3 Издвајање карактера стринга

Постоје три методе за издвајање карактера стринга:

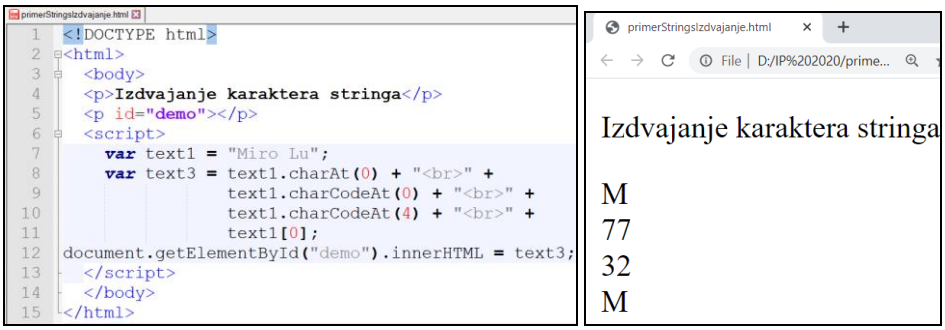
- Метода `charAt(position)`.
- Метода `charCodeAt(position)`.
- Приступ особини, *property access* [ ] .

Метода `charAt()` враћа карактер у стрингу према аргументу који представља индекс (позицију) у стрингу.

Метода `charCodeAt()` враћа *unicode* карактера у стрингу према аргументу који представља индекс (позицију) у стрингу. Метода враћа *UTF*-16 код који је целобројна вредност из опсега од 0 до 65535.

За разлику од ове две методе где се у обичним заградама задаје вредност параметра као аргумент функције, трећа метода користи угласте заграде између којих је аргументу који представља индекс (позицију) у стрингу, као да се ради о низу. На пример:

```
var text1 = "Miro Lu";
text1.charAt(0);          // vraca M
text1.charCodeAt(0);      // vraca 77
text1.charCodeAt(4);      // vraca 32
text1[0];                 // vraca M
```



Слика 3.1.41 Пример три методе за издвајање карактера стринга.

На слици 3.1.41 је показано како се издваја први карактер стринга који има индекс 0, а то је слово M. Његов *unicode* карактер је 77. За бланко карактер се добија *unicode* карактер 32.

Иако трећа метода даје исти резултат у овом примеру као и прва метода, а то је слово M, у неким прегледачима се могу појавити неочекивани резултати. Иако угласте заграде личе као да се ради о низу, стринг није низ. На пример, ако не постоји карактер на позицији коју одређује аргумент, угласте заграде `[]` ће вратити *undefined*, док ће `charAt()` вратити

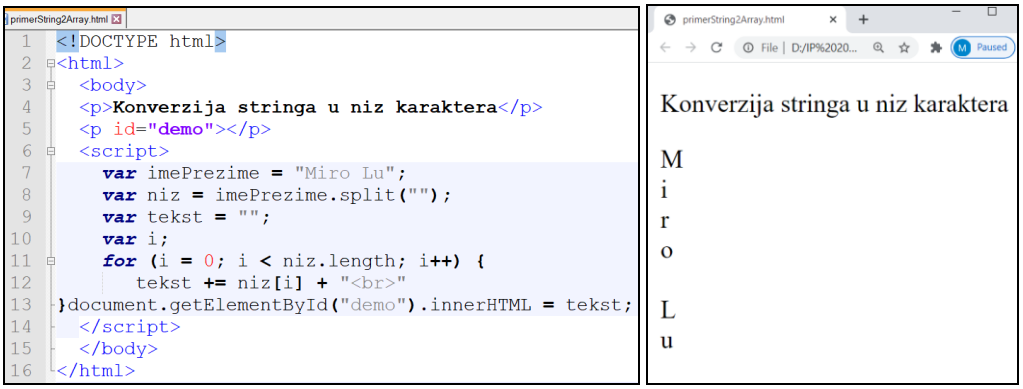
празан стринг. Друга битна разлика између треће методе и низа јесте та да се може прочитати вредност на позицији коју одређује аргумент, али да се не може променити вредност у стрингу доделом вредности. На пример, `text1[0]= "A"`, неће променити вредност првог карактера тако да од `"Miro Lu"` постане `"Airo Lu"`. Међутим, неће бити јављена ни грешка, иако у ствари команда неће урадити то што је програмер написао.

Ако постоји потреба да се са стринговима ради као са низовима, тада прво треба конвертовати стринг у низ, применити операције на низовима, и поново вратити у стринг ако треба да крајњи резултат буде стринг.

Метода `split()` се користи за конвертовање стринга у низ тако што се дефинишу сепаратори као зарези (*commas*), бланко карактери (*spaces*) или вертикалне црте (*pipe, vertical bars*):

```
var txt = "a,b,c,d,e";   // string
txt.split(",");          // separator zarez
txt.split(" ");          // separator blanko znak
txt.split("|");          // separator vertikalna crta, pipe
```

Ако се сепаратор изостави, биће враћен цео стринг као први елемент низа који има индекс 0. Ако је сепаратор празан стринг, двоструки наводници без карактера између њих, "", сваки карактер стринга постаће елемент низа, тако да сваки елемент низа садржи по један карактер.



Слика 3.1.42 Пример конверзије стринга и низ.

На слици 3.1.42 приказана је конверзија стринга у низ и приказ сваког елемента низа у новом реду.

3.1.19 *JavaScript* бројеви

JavaScript има само један тип бројева. Бројеви могу бити записани са децималном тачком (*floating-point*, за рационалне бројеве у српском језику се користи децимални зарез) и без децималне тачке (целобројни, *integers*), на пример:

```
var x1 = 34.00;       // Sa decimalnom tačkom, racionalni broj
var x2 = 34;          // kao celobrojni
var y = 123e5;        // 12300000, kao celobrojni
var y = 123 * 100000; // 12300000, kao celobrojni
var z = 123e-5;       // 0.00123, racionalni broj
var z = 123 / 100000; // 0.00123, racionalni broj
```

За разлику од других програмских језика, *JavaScript* не дефинише различите типове променљивих за бројеве и користи исти стандард *international IEEE* 754 *standard*. JavaScript бројеви се складиште као бројеви двоструке тачности (*double precision*) са покретном децималном тачком (*floating point numbers*). Бројеви се памте у бинарном запису са 64 бита, при чему се 52 бита, од 0 до 51, користи за мантису (цифре којима се памти значајни део броја), док се са 11 бита, од 52 до 62, памти експоненцијални део тако да је број производ мантисе и $10^{\text{експонент}}$; један бит, бит 63, се користи за означавање знака броја (да ли је позитиван или негативан); то је укупно 64 бита (52+11+1 бит).

У неким програмским језицима се користе и типови *short* и *long*, али то није случај са *JavaScript*-ом.

За разлику од других програмских језика где се целобројне представљају са 63 бита, зато што се не користи експоненцијални део, у *JavaScript*-у се користи 52 бита за представљање целобројних. Стога је и тачност операција са целобројним на 15 цифара. На пример:

```
var x = 999999999999999;   // x je   999999999999999
var y = 9999999999999999;  // y je 10000000000000000
var z = y - x;             // z je  9000000000000001
```

Из овог примера се види да `x` садржи тачну вредност, али да је `y` заокружен на већу вредност. Применом оператора за одузимање, добија се приближна вредност `9000000000000001`, а не тачна која би требало да буде `9000000000000000`. У другим програмским језицима оператори са целобројним увек дају тачан резултат ако су бројеви у опсегу представљања, који има већи број цифара за целобројне у односу на рационалне бројеве. Због заокруживања резултата може се појавити грешка у финалном резултата услед заокруживања, па је неопходно да се прво ураде све операције са најмањим вредностима а тек на крају са највећом целобројном. Треба увек имати на уму да је број цифара које су тачне, и у раду са целобројним, највише 15 цифара.

Максималан број децимала је 17, али операције у аритметици са покретном тачком (на српском са покретним зарезом) не дају увек тачну вредност, на пример:

```
var x = 0.2 + 0.1;         // x je 0.30000000000000004
```

Као што у децималном запису број 1/3 не може да се представи коначним бројем децималних цифара, тако ни број 1/5 или 0.2 не може да се представи коначним бројем бита. Зато је у овом примеру дошло до заокруживање на коначан број бита којима се представља број који је у коду представљен у децималном запису. Уколико у програмском коду постоји већи број оваквих операција, може доћи до акумулирања грешке која као коначан резултат може да постане значајна. Статистички гледано, доћи ће до заокруживања на већи или мањи број у бинарној представи, па ће грешка конвергирати ка 0; међутим, као максимална могућа грешка услед заокруживања увек на већу или увек на мању вредност, грешка може значајно да промени финални резултат.

Уобичајени поступак да се избегне грешка при заокруживању услед представљања коначним бројем бита јесте да се сви рационални бројеви помноже са 10 на максимални број децималних цифара, тако да сви бројеви постану целобројни у децималном запису; ако није дошло до прекорачења у броју цифара, сви бројеви ће бити тачно представљени и у бинарном запису, а коначан резултат се подели са 10 на максимални број децималних цифара, када се не појављује грешка услед заокруживања. На пример:

```
var x = (0.2 * 10 + 0.1 * 10) / 10;       // x je 0.3
```

3.1.19.1 Оператор + за бројеве и стрингове

JavaScript користи оператор + за сабирање бројева и за конкатенацију стрингова. Ако је макар један од два операнда стринг, најпре ће један операнд типа број бити конвертован у стринг и биће извршена конкатенација стрингова.

Редослед извршавања оператора истог нивоа извршава се са лева у десно. Ако је постоје два оператора + тада се најпре изврши операција на прва два операнда па затим резултат прве операције користи као аргумент наредне операције са оператором +. На пример:

```
var x = 10;         // x je broj 10
var y = 20;         // y je broj 20
var z = "40";       // z je string 40
var w = x + y + z; // w je string 3040
```

Први оператор +, има два операнда која су оба типа број и даје резултат сабирања два броја. Други операнд има је типа стринг, па се зато резултат 30 типа број конвертује у стринг и други оператор извршава конкатенацију два стринга.

У овом примеру се види да променљива може да има само цифре а да тип може да буде број или стринг; у зависности од тога да ли су операнди типа број или типа стринг могу да се добију различити резултати. Ово важи само када је оператор +.

Када се користе остали аритметички оператори (-, *, /) *JavaScript* примењује другачија правила. Ако је један или су оба операнда написани као бројеви, без обзира да ли су типа број или стринг, сви стрингови ће бити конвертовани у тип број и операције ће се извршити као да су оба операнда типа број.

Ако је један од операнада типа стринг који не може да се конвертује у тип број, тада ће аритметички оператори -, * и / произвести резултат који је NaN.

3.1.19.2 Специјалне вредности и резервисане речи, Infinity и NaN

JavaScript користи резервисану реч NaN која указује да вредност није типа број у ситуацијама када извршавање неке операције не може да се представи бројем. На пример:

```
var x = 100 / "Miro";  // x je NaN (Not a Number)
var y = 0 / 0;         // y je NaN (Not a Number)
var v = 0 * "Miro";    // v je NaN (Not a Number)
```

У првом случају, x није број зато што се не зна која је вредност када се број подели са стрингом који није састављен од цифара. У другом случају, y није број зато што се за количник 0 са 0 не зна који је то број. У трећем случају, v није број зато што се за производ 0 и нечега што се не зна шта је, не зна се који је то број. Производ 0 и

бесконачне вредности такође генерише резултат NaN. Ако је један од операнада за аритметичке операције NaN, тада је и резултат NaN. Када се користи оператор +, а један од операнада је стринг док је онај други NaN, тада се NaN конвертује у стринг NaN, и примењује се правило конкатенације.

Иако NaN значи (Not a Number), резултат испитивања типа података даје да је то број (number):

```
var x = NaN;          // x je NaN
typeof NaN;           // vraca number
typeof(typeof NaN);   // vraca  string
```

Међутим, ако се испитује са typeof(typeof NaN), тада се добија да је резултат стринг string. Другим речима, за x = NaN тип података је број, иако у директном преводу NaN значи да то није број (Not a Number). Најбоље је да се интерпретира да је NaN број за који се не зна која је вредност, па не може да се представи цифрама већ се користи резервисана ознака.

Специјалне вредности броја су Infinity и -Infinity. Ове две вредности се добијају када је резултат већи од највећег броја који може да се представи цифрама (Infinity), или је мањи од најмањег негативног броја (-Infinity). Дељење позитивног броја са нулом генерише Infinity, а дељење негативног броја са нулом генерише -Infinity. Количник две вредности које су Infinity дају резултат NaN. Међутим количник било ког броја са Infinity даје резултат 0.

За разлику од резервисане речи NaN, која као операнд аритметичке операције увек даје резултата NaN, дељење са Infinity даје резултат 0 што значи да то није резервисана реч већ стварно број који има вредност већу од највеће која може да се представи цифрама. Тестирање типа података са typeof Infinity; показује да је број.

3.1.19.3 Бројеви за другачије основе

Подразумева се да *JavaScript* користи цифре за основу 10 ако је прва цифра ненулта. Ако нумеричка константа почиње са 0x тада *JavaScript* подразумева да је број у хексадецималном запису (основа 16). Неки прегледачи интерпретирају нумеричке константе да су у окталном запису ако је прва цифра 0. Запис бројева са основом бројног система се назива и запис бројева са базом која у *JavaScript*-у може да буде од 2 до 36.

Рационални бројеви се записују цифрама које су од 0 до основа бројног система мање 1, тако што се се израчунава збир производа цифре и основе на експонент према позицији записа, на пример број записан као

$$c_3c_2c_1c_0.c_{-1}c_{-2}$$

има вредност

$$broj = c_3\times B^3 + c_2\times B^2 + c_1\times B^1 + c_0\times B^0 + c_{-1}\times B^{-1} + c_{-2}\times B^{-2}$$

У декадном систему цифре **c** су неки од бројева {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} а основа **B** је 10.

$$broj = c_3\times 1000 + c_2\times 100 + c_1\times 10 + c_0\times 1 + c_{-1}\times 0.1 + c_{-2}\times 0.01$$

У хексадецималном систему цифре **c** су неки од бројева {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F} а основа **B** је 16, при чему децимална вредност за A је 10, за B је 11, за C је 12, за D је 13, за E је 14 и за F је 15, на пример

$$broj = c_3 \times 16^3 + c_2 \times 16^2 + c_1 \times 16 + c_0 \times 1 + c_{-1} \times (1/16) + c_{-2} \times (1/16^2)$$

У окталном систему цифре **c** су неки од бројева {0, 1, 2, 3, 4, 5, 6, 7} а основа **B** је 8,

$$broj = c_3 \times 8^3 + c_2 \times 8^2 + c_1 \times 8 + c_0 \times 1 + c_{-1} \times (1/8) + c_{-2} \times (1/8^2)$$

У бинарном систему цифре **c** су неки од бројева {0, 1} а основа **B** је 2,

$$broj = c_3 \times 8 + c_2 \times 4 + c_1 \times 2 + c_0 \times 1 + c_{-1} \times 0.5 + c_{-2} \times 0.25$$

Да би се разликовали бројеви у различитим бројним системима, записи у свим другим бројним системима се чувају као стринг, с тим да програмер мора увек да има на уму која је основа бројног система. На пример, број записан као 100101 у различитим бројним системима има различите вредност, па ће и интерпретација записа бити погрешна ако програмер користи запис са погрешним бројним системом.

Конверзија бројева из декадног бројног система у стрингове који садрже представљање бројева у системе са другом основом врши се неком од следећих метода:

```
var myNumber = 32;
myNumber.toString(10);  // vraca 32
myNumber.toString(32);  // vraca 10
myNumber.toString(16);  // vraca 20
myNumber.toString(8);   // vraca 40
myNumber.toString(2);   // vraca 100000
```

Стрингови садрже цифре које су од 0 до 9, и словне ознаке за цифре ако је основа већа од 10. На пример, број 31 у декадном запису биће конвертован у мало латинично слово v за основу 32, где слово v има вредност 31 у запису бројева са основом 10.

3.1.19.4 Бројеви као објекти

Подразумева се да су *JavaScript* бројеви примитиван тип података који се креирају као константе. Међутим, бројеви могу да се дефинишу као објекти када се користи кључна реч new, на пример:

```
var x = 135;             // typeof x vraca number
var y = new Number(135); // typeof y vraca object
```

Не препоручује се да се креирају бројеви као објекти зато што то може да успори извршавање програмског кода. Кључна реч new компликује код, а може да проузрокује и неочекиване резултате. У следећем примеру, иста је вредност за променљиву x типа број и објекта y:

```
var x = 567;
var y = new Number(567);
// (x == y) daje vrednost true zato sto x i y imaju istu vrednost
```

Међутим, у наредном примеру

When using the === operator, equal numbers are not equal, because the === operator expects equality in both type and value.

```
var u = 789;
var v = new Number(789);
// (u === v) daje vrednost false zato sto su u i v razlicitog tipa
```

Још је гора ситуација када су за оператор == оба операнда тупа објекат;

```
var z = new Number(236);
var w = new Number(236);
// (z == w) daje false zato sto objekti z i w ne mogu da se uporedjuju
```

Да се подсетимо, оператор == упоређује вредности, а оператор === упоређује да ли су исти и типови података. Међутим, када су бројеви декларисани као објекти, оператор === враћа вредност `false` зато што објекти не могу да се упоређују.

3.1.20 *JavaScript* особине и методе за бројеве

Тип података бројеви су примитивне вредности које имају вредност специфицирану цифрама и као такви не могу да имају особине и методе као што их имају објекти. Међутим, *JavaScript* има особине и методе које се могу применити и на примитивне вредности као што су стрингови и бројеви зато што *JavaScript* третира примитивне вредности као објекте при извршавању *JavaScript* особина и метода као да су објекти са циљем да се олакша рад са бројевима.

Метода `toString()` враћа број као стринг. На пример,

```
var x = 123;             // promenljiva x tipa broj
x.toString();            // vraca string 123 iz promenljive x
(123).toString();        // vraca string 123 iz konstante 123
(100 + 23).toString();   // vraca string 123 iz izraza (100 + 23)
```

Метода `toString()` конвертује променљиву којој је додељена вредност типа број, константу (*literal*) или израз (*expression*) који је типа број, у податак типа стринг. У суштини метода ради као функција која као улазни аргумент има податак типа број, конвертује податак у стринг и враћа излазни податак типа стринг. Уместо да се после назива функције `toString` аргумент функције записује између обичних заграда `()`, користи се запис са тачком тако да је то што се налази лево од тачке уствари аргумент функције а десно од тачке је назив функције са обичним заградама `toString()`. Због тога се по основу оваквог записа са тачком између аргумента и функције стиче утисак да примитивна вредност има своју методу и да се примитивна вредност третира као објекат. Како се овде ради о запису који личи на методу објекта, увек се мора имати на уму да примитивна вредност ипак није објекат. Наравно, може се декларисати објекат који има исту вредност као и примитивна вредност, али *JavaScript* другачије интерпретира податке и може доћи до изненађујућих резултата, као на пример да оператор ==, који прихвата аргументе типа број, не може да се користи када су оба податка типа објекат без обзира што имају исту вредност:

```
var z = new Number(236);
var w = new Number(236);
// (z == w) daje false zato sto objekti z i w ne mogu da se uporedjuju
```

У математици се користи израз постфиксна нотација (*postfix notation*) у којој операнди претходе оператору. У *JavaScript*-у постоји операнд (податак типа број или типа стринг, на пример `x`) после кога се уместо оператора пише функција (у претходном примеру `toString`), а да би се избегла вишезначност оваквог записа, пише се тачка између операнда и оператора, на пример `x.toString()`. У математици би то значило да се операнд који претходи оператору користи као аргумент оператора, а у *JavaScript*-у да се то што је лево од тачке користи као аргумент функције које је десно од тачке. Овакав запис је идентичан са објектима, где је објекат лево од тачке, а метода десно од тачке, тако да се као резултат добија особина која важи за објекат.

Сличност записа са тачком за примитивне вредности не сме да се погрешно интерпретирати да је примитивна вредност (типа број или типа стринг) у ствари објекат. У *JavaScript*-у примитивна вредност може да се декларише као објекат када се користи кључна реч `new`.

Метода `toExponential()` враћа приближну вредност броја коришћењем експоненцијалне нотације као податак типа стринг. Параметар у загради одређује број цифара после децималне тачке; испред децималне тачке се издваја водећа цифра промењиве, а после децималне тачке се приказује онолико цифара колико је у загради методе; ако је број цифара који је додељен промењивој мањи од броја цифара промењиве, додају се нуле у десно; ако је број цифара промењиве већи од броја места после децималне тачке, долази до заокруживања на приближну вредност. На пример

```
var x = 0.8656;         // promenljiva x tipa broj
x.toExponential();      // vraca string 8.656e-1
x.toExponential(2);     // vraca string 8.66e-1
x.toExponential(4);     // vraca string 8.6560e-1
x.toExponential(6);     // vraca string 8.656000e-1
```

У овом примеру је водећа ненулта цифра је 8, која се пише пре децималне тачке. Остале цифре се пишу редом од децималне тачке у десно. Ако је број разломљених цифара већи од аргумента у загради, тада се до пуног броја цифара допуњују нуле. Када је број места за разломљени део мањи од броја цифара, врши се заокруживање; на пример за `x = 0.8656`, враћа број заокружен на већу вредност као стринг `8.66e-1`, док би за `x = 0.8655`, био враћен број заокружен на мању вредност као стринг `8.65e-1`. Ако није дат аргумент на месту параметра методе, враћају се све цифре промењиве у експоненцијалном формату.

У следећем примеру можемо да проверимо шта се дешава када се промењивој `x` додељује вредност (`x = 0.2`). Када се не одређује број цифара десно од децималне тачке `x.toExponential()`, тада се добија да је то број који је и додељен промењивој `2e-1`.

```
var x = 0.2;            // promenljiva x tipa broj
x.toExponential();      // vraca string 2e-1
x.toExponential(18);    // vraca string 2.00000000000000111e-1
(0.1+0.1+0.1-0.3).toExponential(18); // vraca 5.551115123125782702e-17
```

Када се параметар методе постави на 18 цифара, добија се `2.00000000000000111e-1`. Овиме се потврђује да децимални број који не може да се представи коначним бројем бита бива заокружен на број који није једнак децималном броју а који је додељен промењивој. У већини случајева ово неће бити проблем да апликације исправно раде, али у неким специфичним случајевима могу настати и озбиљне грешке. На пример, ако се

лево од тачке напише израз који три пута сабере број `0.1` и одузме `0.3`, уместо да буде резултат `0`, добија се број који је различит од нуле, `5.551115123125782702e-17`.

Метода `toFixed()` враћа стринг који се састоји од броја као улазног податка са специфицираним бројем децимала. Параметар у загради одређује број цифара десно од децималне тачке, на пример:

```
var x = 0.8656;   // promenljiva x tipa broj
x.toFixed();      // vraca string 1
x.toFixed(0);     // vraca string 1
x.toFixed(2);     // vraca string 0.87
x.toFixed(4);     // vraca string 0.8656
x.toFixed(6);     // vraca string 0.865600
```

Уколико је број цифара десно од децималне тачке мањи од цифара броја, најпре се врши заокруживање (за број лево од децималне тачке `0.8656`: за атрибут `(0)`, или без атрибута `()`, добија се стринг `1`; за атрибут `(2)`, добија се `0.87`; за атрибут `(4)`, добија се `0.8656`) па се тек после тога број конвертује у стринг. Ако се не напише која је вредност параметра методе, подразумева се да је `0`. Ако је број цифара после децималне тачке мањи од вредности параметра у заградама, додају се 0.

За апликације са новцем, где се сви резултати приказују на две децимале, погодно је приказивање резултата методом `toFixed(2)` са атрибутом 2; Међутим за израчунавања је свакако погодније да се користе бројеви са већим бројем децимала да не би дошло до акумулиране грешке услед заокруживања.

Метода `toPrecision()` враћа стринг који се састоји од броја као улазног податка са специфицираним бројем цифара. Параметар у загради одређује број цифара (укупно лево и десно од децималне тачке), при чему се допуњује нулама ако је атрибут већи од броја цифара броја. На пример:

```
var x = 0.8656;          // promenljiva x tipa broj
x.toPrecision();         // vraca string 0.8656
x.toPrecision(2);        // vraca string 0.87
x.toPrecision(4);        // vraca string 0.8656
x.toPrecision(6);        // vraca string 0.865600
```

У примеру треба уочити да се у број цифара које се приказују не укључује нуле лево од прве ненулте цифре. Ако вредност параметра није задата, `()`, тада се у стрингу појављују све цифре броја на који се примењује ова метода. Ако се ова метода примени на рационалне бројеве који не могу да се прикажу тачно у бинарном запису, добија се на пример

```
var x = 0.2;          // promenljiva x tipa broj
x.toPrecision(25);    // vraca string 0.2000000000000000111022302
```

Метода `valueOf()` враћа број као податак типа број. На пример

```
var x = 123;             // promenljiva x tipa broj
x.valueOf();             // vraca broj 123 iz promenljive x
(123).valueOf();         // vraca broj 123 iz konstante 123
(100 + 23).valueOf();    // vraca broj 123 iz izraza (100 + 23)
```

У претходном примеру, променљива (`var x`) је декларисана као податак типа број. Метода `valueOf()` враћа број, што нема пуно смисла зато што је је променљива већ декларисана као број. Међутим, ако је променљива декларисана као објекат коришћењем кључних речи са `new Number(123)`, тада метода `valueOf()` враћа из објекта податак типа број, на пример:

```
var x = new Number(123); // promenljiva x tipa objekat
x.valueOf();   // vraca broj 123 iz promenljive x tipa objekat
```

У *JavaScript*-у за тестирање типа података се користи `typeof` који враћа резултат `number` ако се тестира примитивна вредност, а враћа резултат *object* ако се тестира објекат. Намена методе `valueOf()` је за интерну употребу у *JavaScript*-у за конверзију броја декларисаног као објекат у примитивну вредност типа број. Све врсте *JavaScript* података могу да користе методе `valueOf()` и `toString()`. Треба увек имати на уму да променљиве декларисане као бројеви и стрингови нису објекти, већ да је запис са тачком врста постфиксне нотације за примитивни тип података и функцију, где број или стринг претходи тачки а функција је после тачке, па запис само личи на методу као да је податак типа објекат.

3.1.20.1 Конверзија променљивих у бројеве

За конверзију променљивих у бројеве могу да се користе три *JavaScript* методе:

1. `Number()` метода
2. `parseInt()` метода
3. `parseFloat()` метода

Ове три методе су глобалне *JavaScript* методе које могу да се користе за све типове *JavaScript* података. Када се ради са бројевима, најзначајније методе су у следећој табели:

Метода	Опис
`Number()`	Враћа број конверзијом од аргумента.
`parseFloat()`	Парсира (рашчлањује) аргумент и враћа број са покретном тачком
`parseInt()`	Парсира (рашчлањује) аргумент и враћа целобројну вредност броја

За разлику од претходних метода наведених у овом поглављу, три наведене методе су глобалне методе, а не само методе за потребе бројева.

У рачунарству рашчлањивање (парсирање, *parsing*) је део интерпретатора у коме се прихвата улазни текст и претвара у облик који је погодан за даљу обраду. У рачунарству се користи и термин *parser* за скриптни језик који се употребљава за веб апликације и скрипте на серверској страни.

Метода `Number()` може да се користи за конверзију променљивих у податке типа број. На пример:

```
Number(true);          // vraca broj 1
Number(false);         // vraca broj 0
Number("10");          // vraca broj 10
Number("  10");        // vraca broj 10
Number("10  ");        // vraca broj 10
```

```
Number(" 10  ");        // vraca broj 10
Number("10.33");        // vraca broj 10.33
Number(" 8.656e-1 ");   // vraca broj 0.8656
Number("10,33");        // vraca NaN tipa broj
Number("10 33");        // vraca NaN tipa broj
Number("John");         // vraca NaN tipa broj
```

Метода `Number()` конвертује логичке вредности у број 0 или 1. Стрингове који садрже цифре, ова метода конвертује у број, с тим да бланко карактери се не појављују као део враћене вредности. Ако између цифара постоји децимална тачка, број се конвертује у рационалан број. Ако стринг садржи експоненцијални запис броја, као што је на пример `"8.656e-1"`, биће враћен рационални број `0.8656`. Ако се уместо децималне тачке појављује зарез или бланко карактер као сепаратор целобројног и разломљеног дела броја, резултат ће бити `NaN`, означавајући да је тип података број коме није позната вредност. Ако осим бројева стринг садржи и слова, резултат је `NaN`. У свим ситуацијама када се стринг не може конвертовати у број, резултат је `NaN`.

Када се користи објекат за датум, метода `Number()`конвертује вредности у број исказан у јединицама милисекунде рачунајући од 1.1.1970. На пример, за објекат дефинисан са `Date("...")`,

```
Number(new Date("2020-08-05")); // vraca broj 1596585600000 u ms
```

Метода `parseInt()` расчлањује текст који је аргумент методе и извлачи први цели број који може да препозна, на пример:

```
parseInt("10");         // vraca broj 10
parseInt("10.33");      // vraca broj 10
parseInt("8.656e-1");   // vraca broj 8
parseInt("10 20 30");   // vraca broj 10
parseInt("10 years");   // vraca broj 10
parseInt("years 10");   // vraca NaN tipa broj
parseInt(new Date("2020-08-05"))  // vraca NaN
```

Ако стринг садржи цео број, метода ће вратити тај број. Ако постоји сепаратор од остатка текста, у виду тачке, бланко карактера, двотачке или било ког слова, метода враћа тај број. Ако `parseInt()` не препозна неки број као први карактер, односно ако расчлањивањем не препозна број, враћа се као резултат `NaN`. Ако је аргумент објекат, прегледач неће приказати резултат. Треба уочити да за експоненцијални запис као стринг, `"8.656e-1"`, метода `parseInt()` препознаје водећу цифру 8 као целобројну, а не конвертује стринг у број и тек после тога издваја водећи број.

Метода `parseFloat()` расчлањује текст који је аргумент методе и извлачи први број који може да препозна као рационални (ако има децималну тачку) или целобројни, на пример:

```
parseFloat("10");         // vraca broj 10
parseFloat("10.33");      // vraca broj 10.33
parseFloat("8.656e-1");   // vraca broj 0.8656
parseFloat("10 20 30");   // vraca broj 10
parseFloat("10 years");   // vraca broj 10
parseFloat("years 10");   // vraca NaN tipa broj
```

Уколико не може да препозна рационални број (који има тачку између цифара) или целобројни, тада метода враћа `NaN`. Треба уочити да за експоненцијални запис као стринг, `"8.656e-1"`, метода `parseFloat()` препознаје цео рационални број.

3.1.20.2 Особине бројеве

У табели су дате неке од важнијих особина које се користе за *JavaScript* бројеве:

Метода	Опис
`MAX_VALUE`	Враћа највећи број који је могућ у *JavaScript*-у
`MIN_VALUE`	Враћа најмањи број који је могућ у *JavaScript*-у
`POSITIVE_INFINITY`	Враћа бесконачно (*infinity*) када се деси прекорачење (*overflow*)
`NEGATIVE_INFINITY`	Враћа негативно бесконачно (-*infinity*) када се деси прекорачење
`NaN`	Представља број за који се не зна која је вредност (*Not-a-Number*)

На пример:

```
var x = Number.MAX_VALUE; // vraca 1.7976931348623157e+308
var x = Number.MIN_VALUE; // vraca 5e-324
var x = Number.POSITIVE_INFINITY; // vraca Infinity
var x = 1 / 0;                    // vraca Infinity
var x = Number.NEGATIVE_INFINITY; // vraca -Infinity
var x = -1 / 0;                   // vraca -Infinity
var x = Number.NaN;               // vraca NaN
```

Било која аритметичка операција са `NaN` враћа резултат `NaN`. У *JavaScript*-у, аритметичке операције `-`, `*` и `/`) где је један операнд број а други стринг који није састављен од цифара, као резултат враћају `NaN`.

Особине бројева се могу применити само ако се примењују са `Number`. У случају да се примене на промењљиве, биће враћен резултат *undefined*. На пример:

```
var x = 6;
var y = x.MAX_VALUE;    // y postaje undefined
```

Овај пример показује да особине бројева захтевају да се леве стране тачке постоји `Number`.

3.1.21 Питања и задаци за вежбу

1. Навести разлоге за коришћење *JavaScript*-а.
2. Шта значи појам да је *JavaScript* објектно базиран?
3. Да ли је могуће модуларно програмирање са *JavaScript*-ом?
4. Шта значи да је *JavaScript* интегрисан са *HTML*-ом?
5. Који програм се користи за приказ рада кода написаног у *JavaScript*-у?
6. Да ли је *JavaScript* код робустан као *HTML*?
7. Како се ради промена *HTML* садржаја коришћењем *JavaScript*-а?
8. Написати пример приказа садржаја у прегледачу коришћењем метода `getElementById()`.
9. Како се може да промени вредност атрибута у *HTML* документу коришћењем *JavaScript*-а?
10. Како се убацује *JavaScript* код у *HTML* документ?
11. Како се и зашто користи `document.write()`?

12. Шта су у *JavaScript*-у наредбе, вредности, оператори, изрази, кључне речи и коментари? Написати једноставан пример.
13. Да ли постоји разлика у употреби бланко карактера између *HTML* и *JavaScript*-а? Објаснити.
14. Написати пример најједноставније *JavaScript* синтаксе.
15. Како се и зашто користе *JavaScript* коментари?
16. Написати примере за најмање три типа података у *JavaScript*-у.
17. Које врсте оператора постоје?
18. Колико оператор може да има операнда у *JavaScript*-у? Објаснити.
19. Шта су *JavaScript* функције и како се користе? Колико параметара може да има једна функција?
20. Како се декларишу *JavaScript* објекти?
21. Која је улога *Escape* карактера у *JavaScript* стринговима?
22. Да ли *JavaScript* стринг може да се декларише као објекат? Које су предности?
23. Да ли *JavaScript* број може да се декларише као објекат? Који су недостаци?
24. Како се пронађи стринга у *JavaScript* стринговима?
25. Како се издваја карактера из стринга?
26. У ком формату у *JavaScript*-у се записују рационални бројеви, а у ком целобројни?
27. Објаснити функционисање *JavaScript* оператора `+` за бројеве и стрингове?
28. Шта значи `NaN` и ког је типа?
29. Објаснити зашто `x.toPrecision(0.2);` даје резултат 0.20000000000000000111022302.
30. Описати једну од *JavaScript* методе за конверзију променљивих у бројеве.
31. Написати пример када се добија резултат `Infinity`.
32. Објаснити када се и зашто добија за променљиву да је `undefined`.

3.2 Сложени типови *JavaScript* података

Резиме

Циљ овог поглавља је да се студенти оспособе за сложеније типове *JavaScript* података као што су низови и објекти за датум и време, а посебно за Get Date и Set Date методе.

Увод у сложене типове *JavaScript* података

JavaScript има примитивне вредности као што су бројеви и стрингови, као што је детаљно објашњено у претходном поглављу. *JavaScript* има могућност да ради и са сложеним типовима података а то је тема овог поглавља.

3.2.1 *JavaScript* низови

JavaScript низови (*arrays*) се користе да се у једној променљивој запамти већи број вредности. *JavaScript* низови се записују у угластим заградама. Елементи низова су раздвојени зарезима. Следећи пример декларише низ који има назив `prijatelji` и садржи три стринга са именима пријатеља, `Miro`, `Ana`, `Branka`:

```
var prijatelji = ["Miro", "Ana", "Branka"];
```

Првом елементу низа се придружује индекс 0 који се означава са [0], другом се придружује индекс 1 који се означава са [1], и тако даље, сваки наредни има индекс већи за 1 од претходног.

За разлику од низова, *JavaScript* објекти се пишу са витичастим заградама {}. Особине објеката се пишу као парови (између којих је знак за двотачку, назив:вредност) а парови су раздвојени зарезима као и код низова.

Разлог за генерисање промењљивих као низови може се најлакше објаснити на следећем примеру. Дефинишимо три промењљиве типа стринг, на пример,

```
var prijatelj1 = "Miro";
var prijatelj2 = "Ana";
var prijatelj3 = "Branka";
```

Овакав запис је погодан све дотле док је број промењљивих мали, али постаје веома компликовано да радимо са свим стринговима када је број промењљивих неколико стотина. Немогуће је да програмер има у глави који је број додељен којем стрингу, да проналази неке карактеристике и на основу њих генерише друге резултате, као што је сортирање, брисање или убацивање нове промењљиве. Да би се олакшале манипулације са већим бројем података, пожељно је да се све налазе под истим називом промењљиве, а то је промењљива која је низ. Сви подаци се налазе као елементи низа, а редни број позиције елемента у низу је додатни број који смо у претходном примеру ручно додавали. Уместо декларисаних промењљивих `prijatelj1`, `prijatelj2`, `prijatelj3`, декларише се једна промењљива `prijatelji`, при чему је на првој позицији `prijatelj1`, на другој позицији `prijatelj2`, и на позицији 3 је `prijatelj3`, односно

```
var prijatelji = ["Miro", "Ana", "Branka"];
```

Бланко карактери пре и после стринга наведеног између знакова навода немају утицај на доделу вредности низа. Такође, после зареза се може прећи у нови ред без утицаја на садржај низа, на пример:

```
var prijatelji = [
      "Miro",
      "Ana",
      "Branka"
    ];
```

Низови су специјалан тип објекта. *JavaScript* оператор `typeof` враћа "object" за низ. Обзиром да је низ специјалан облик објекта, није пожељно да се декларише коришћењем кључне речи `new`, иако се низ може креирати и као објекат:

```
var prijatelji = new Array("Miro", "Ana", "Branka");
```

Иако је у функционалном смислу исто да ли ће се низ генерисати без или са `new`, због једноставности, читљивости кода и веће брзине извршавања боље је да се промењљива низ креира без кључне речи `new`.

Елементу низа се приступа преко целобројног индекса при чему првом елементу одговара индекс 0, другом елементу низа одговара индекс 1, и тако редом. На пример:

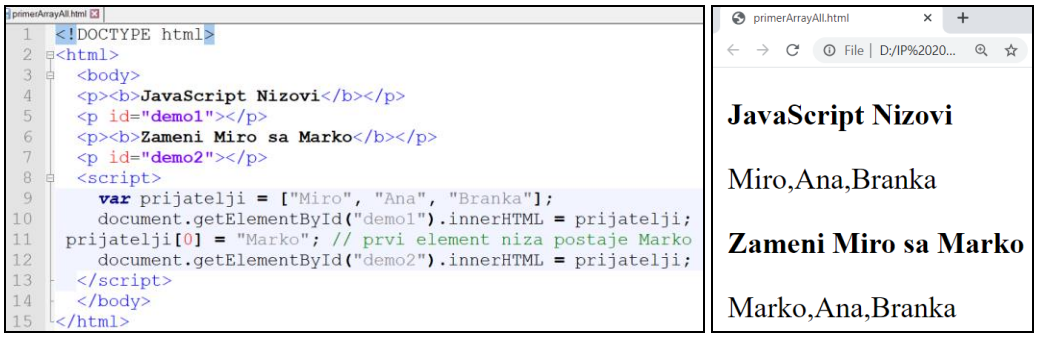
```
var ime1 = prijatelji[0];  // rezultat je string Miro
var ime2 = prijatelji[1];  // rezultat je string Ana
```

```
var ime3 = prijatelji[2];  // rezultat je string Branka
```

Промена вредности елемента низа може да се уради приступом по индексу, на пример:

```
var prijatelji = ["Miro", "Ana", "Branka"];
prijatelji[0] = "Marko";  // prvi element niza postaje Marko
```

Свим елементима низа се приступа називом промењиве без специфицирања индекса:



Слика 3.2.1 Промена вредности првог елемента низа и приказ свих чланова низа.

На слици 3.2.1 приказано је у прегледачу како се мења вредност првог елемента низа и приказују сви чланови низа у прегледачу.

Док се елементу низа приступа коришћењем индекса између угластих заграда, објекту се приступа по називу особине, на пример:

```
var osoba = {ime:"Miro", prezime:"Lu", godine:64};
var ime1 = osoba.ime;  // rezultat je string Miro
```

Елементи низа могу бити различитог типа, на пример први елемент може да буде датум, други да буде функција а трећи елемент низа може да буде нови низ.

У неким верзијама прегледача може да се користи зарез после последњег елемента низа, на пример, исти резултат се добија са оба следећа записа и дужина низа је 3 у оба случаја:

```
var prijatelji = ["Miro", "Branka", "Ana"];
var prijatelji = new Array("Miro", "Ana", "Branka",);
```

Претходни пример не функционише у свим прегледачима.

У програмирању се често користе са асоцијативним именима уместо индекса, и овакви низови се називају асоцијативни низови (*associative arrays, hashes*). *JavaScript* не подржава низове са именованим индексима.

JavaScript не подржава низове са именованим индексима. *JavaScript* низови увек користе нумерисане индексе. Да би се користили именовани индекси, потребно је дефинисати низ као објекат у угластим заградама, где је испред двотачке асоцијативни назив, а одмах после двотачке је елемент низа, на пример

```
var osoba = {ime:"Miro", prezime:"Lu", godine:64};
```

Треба уочити да у овом примеру није коришћена кључна реч new. Треба бити опрезан за коришћењем асоцијативних низова зато што могу као резултат да дају неочекиване вредности.

```
var osoba = {ime:"Miro", prezime:"Lu", godine:64};
var ime1 = osoba["ime"];  // rezultat je string Miro
```

У овом примеру, иако није коришћена кључна реч new, а уместо угластих заграда је коришћена витичаста заграда као за објекте, *JavaScript* интерпретира овакав податак као објекат и у угластим заградама користи именовани индекс да би се добила вредност из низа. Важно је напоменути да вредности објекта може да се приступи са `osoba.ime` или `osoba["ime"]`.

Ако је низ генерисан као објекат са именованим индексима, тада се не могу користити бројеви као индекси.

Иако низ није објекат, већ специјалан тип објекта, елемент низа може да буде регуларан објекат.

Као што особине и методе могу да се придруже примитивним вредностима као што су константе и стрингови, тако се особине и методе могу придружити и низовима. Еквивалентно постфиксној нотацији, у којој операнди претходе оператору, и у случају низа као операнда може се користити постфиксна нотација тако што после назива низа следи тачка као сепаратор и затим особина или метода уместо операнда. Формално посматрајући запис `називНиза.особинаНиза` добија се утисак да је у питању објекат иако се ради о низу, на пример ако уместо `називНиза` пише `prijatelji` а уместо `особинаНиза` пише `length`:

```
var prijatelji = ["Miro", "Branka", "Ana"]; // dodela stringova nizu
var x = prijatelji.length;   // vraca broj elemenata niza, 3
var y = prijatelji.sort();   // sortira niz
```

У трећој линији кода је уместо особине коришћена метода која има пар обичних заграда да би се метода разликовала од особине.

У претходном примеру коришћена је особина низа `length` која враћа број елемената низа. Број елемената низа је за 1 већа од највећег индекса низа зато што је први индекс низа 0, а последњи индекс је за један мањи од дужине низа, на пример:

```
var prijatelji = ["Miro", "Branka", "Ana"]; // dodela stringova nizu
var duzinaNiza = prijatelji.length;         // vraca 3
var prviElement = prijatelji[0];            // vraca string Miro
var poslednjiElement = prijatelji[prijatelji.length - 1]; // vraca Ana
```

При додели новог елемента низа, може се доделити и са индексом који је већи од највећег постојећег индекса у низу. У примеру који следи, додат је стринг са индексом 5, док је највећи индекс у низу био 2; због тога је повећана дужина низа са 3 на 6, а последњи елемент је онај који је додат са индексом 5:

```
var prijatelji = ["Miro", "Branka", "Ana"]; // dodela stringova nizu
var prijatelji[5] = "Ema";      // dodela stringa nizu sa indeksom 5
// vraca niz sa elementima  Miro, Branka, Ana, , , Ema
```

```
var duzinaNiza = prijatelji.length;         // vraca 6
var prviElement = prijatelji[0];            // vraca string Miro
var poslednjiElement = prijatelji[prijatelji.length - 1]; // vraca Ema
```

Додавање елемената са већим индексом од највећег у постојећем низу може да креира неидентификоване "рупе" у низу (*undefined* "*holes*"). Да би се избегло убацивање елемената чија се вредност не зна, уместо уношења броја као индекса где треба да се дода нови елемент, боље је да се користи особина `.length` и тада се не генеришу елементи низа непознате вредности. На пример:

```
var prijatelji = ["Miro", "Branka", "Ana"]; // dodela stringova nizu
var prijatelji[prijatelji.length] = "Ema";  // dodela stringa nizu
// vraca niz sa elementima  Miro, Branka, Ana, Ema
var duzinaNiza = prijatelji.length;         // vraca 4
var prviElement = prijatelji[0];            // vraca string Miro
var poslednjiElement = prijatelji[prijatelji.length - 1]; // vraca Ema
```

Уместо да се приступа елементима низа преко индекса, или да се додају нови елементи низа специфицирањем индекса у коду, боље је да се користе петље, а овај поступак на енглеском се назива *Looping Array Elements*. Следи пример `for` петље (`for` *loop*):

```
var prijatelji, tekst, pLen, i;   // rezervisanje naziva promenljivih
prijatelji = ["Miro", "Branka", "Ana"]; // dodela stringova nizu
pLen = prijatelji.length;         // dodeljuje broj elemenata niza

tekst = "<ul>";                   // pocetak html ulancane liste
for (i = 0; i < pLen; i++) { // od nultog indeksa, sa korakom 1 do pLen
  tekst += "<li>" + prijatelji[i] + "</li>"; // dodaj element u listu
}                           // kraj bloka za petlju
tekst += "</ul>";           // kraj html ulancane liste
```

У прегледачу ће бити приказана уланчана листа са три елемента која су имена типа стрингови

- `Miro`
- `Branka`
- `Ana`

Петља се користи да би се генерисао *html* запис текста за уланчану листу у којој се налазе сви елементи низа. Пре петље се додаје део стринга за почетак уланчане листе `"<ul>`, а затим се пролазом кроз `for` петљу, тексту за приказ у прегледачу додаје по један елемент низа почев од првог (индекс 0), а завршава се са максималним индексом који је за 1 мањи од дужине низа `pLen`. Аутоматизацијом генерисање текста за приказ у прегледачу избегавају се грешке које би настале када би се појединачно генерисао текст, и овакав код не зависи од дужине низа.

Уместо претходног кода, може да се користи функција `Array.forEach()`:

```
var prijatelji, tekst;
prijatelji = ["Miro", "Branka", "Ana"]; // dodela stringova nizu

tekst = "<ul>";                  // pocetak html ulancane liste
prijatelji.forEach(myFunction);  // dodavanje elemenata u listu
tekst += "</ul>";                // kraj html ulancane liste
```

```
function myFunction(value) {
 tekst += "<li>" + value + "</li>";
}
```

За сваки елемент низа се позива `Array.forEach()`, који у текст који треба прегледач да прикаже у параграфу додаје елементе листе и вредност низа. Функција `Array.forEach()` не ради у свим прегледачима.

Метода `push()` може да се користи за додавање елемента постојећем низу.

```
var prijatelji = ["Miro", "Branka", "Ana"]; // dodela stringova nizu
prijatelji.push("Ema");    // dodavanje novog elementa Ema
// prijatelji postaje niz Miro,Branka,Ana,Ema
```

Други начин да се дода елемент низа јесте да се користи дужина низа као индекс елемента који се додаје, на пример:

```
var prijatelji = ["Miro", "Branka", "Ana"]; // dodela stringova nizu
prijatelji[prijatelji.length] = "Ema"; // dodavanje novog elementa Ema
// prijatelji postaje niz Miro,Branka,Ana,Ema
```

Обзиром да коришћење `typeof` враћа тип података да је објекат може да се користи метода `Array.isArray()` за одређивање да ли је променљива низ:

```
var prijatelji = ["Miro", "Branka", "Ana"]; // dodela stringova nizu
Array.isArray(fruits);                      // vraca true
```

У неким верзијама прегледача ова метода не функционише. Други начин је да се користи сопствена функција:

```
function isArray(x) {
  return x.constructor.toString().indexOf("Array") > -1;
}
```

Ова функција увек враћа `true` ако је аргумент низ. Прецизније речено, вратиће `true` ако прототип објекта садржи реч `Array`.

Трећи начин да се провери да ли је тип података низ, јесте да се користи оператор `instanceof` који враћа резултат `true` ако је објекат креиран са задатим конструктором:

```
var prijatelji = ["Miro", "Branka", "Ana"];

prijatelji instanceof Array;   // vraca true
```

У наредном делу биће размотрене методе које важе за низове.

3.2.1.1 Методе за низова

Низови су по типу објекти, али нису декларисани у паровима као објекти у запису са двотачком, назив:метода, па нису асоцијативни низови, већ се елементима приступа по бројној вредности индекса. За разне манипулације са низовима у *JavaScript*-у постоје методе којима се ефикасно обављају разне трансформације података погодне за приказ у прегледачима.

JavaScript метода `toString()` конвертује низ у стрингове раздвојене зарезима (*comma separated array values*).

```
var prijatelji = ["Miro", "Branka", "Ana"]; // dodela stringova nizu
document.getElementById("demo").innerHTML = prijatelji.toString();
// u pregledacu se dobija tekst Miro,Branka,Ana
```

Метода `join()` такође спаја (joins) све елементе низа у стринг који се приказује као текст у прегледачу. Ради слично као и метода `toString()` осим што може да се специфицира сепаратор елемената, на пример ознаком *:

```
var prijatelji = ["Miro", "Branka", "Ana"]; // dodela stringova nizu
document.getElementById("demo").innerHTML = prijatelji.join(" * ");
// u pregledacu se dobija tekst Miro * Branka * Ana
```

Важно је да се запази да ако између елемената текста нема бланко карактера, за дужи део текста неће доћи до прелома текста у прегледачу у више редова. Уколико постоје бланко знаци, и текст не може да стане у једној линији прегледача, текст ће се преломити на два или више реда у прегледачу.

Метода `push()` додаје нови елемент низу после последњег постојећег елемента низа.

```
var prijatelji = ["Miro", "Branka", "Ana"]; // dodela 3 stringa nizu
prijatelji.push("Ema");  //  dodaje novi element nizu ("Ema")
// niz  sada ima 4 elementa ["Miro", "Branka", "Ana", "Ema"]
```

Додавањем новог елемента низа на крај низа повећава се дужина низа за 1. Ако се додавање реализује кликом на дугме, тада ће се после сваког клика дугмета додати нови елемент на крај низа и тиме увећати дужина низа за још 1.

Метода `pop()` избацује из низа последњи елемента низа. Ако се новој промењивој додељује вредност коју враћа метода `pop()`, то је управо вредност елемента који је избачен из низа. На пример:

```
var prijatelji = ["Miro", "Branka", "Ana"]; // dodela 3 stringa nizu
prijatelji.pop();     // izbacuje poslednji element iz prijatelji
prijatelji = ["Miro", "Branka", "Ana"];     // ponovna dodela nizu
var x = prijatelji.pop();  // izbacena vrednost dodeljena x je "Ana"
// niz  prijatelji sada ima 2 elementa ["Miro", "Branka"]
```

Избацивањем елемента низа са краја низа смањује се дужина низа за 1. Ако низ има само један елемент, избацивањем последњег елемента низа добија се да је дужина низа 0. Из низа који нема ни један елемент, не може да се избаци последњи елемент, па дужина низа остаје 0 после примене методе `pop()`.

Метода `shift()` избацује из низа први елемент низа и помера (*shifts*) све остале елементе у низу тако да је индекс сваког елемента низа за 1 мањи у односу на индекс пре примене ове методе. Ова метода ради исто што и метода `pop()`, осим што избацује први а не последњи елемент низа. Код методе `pop()` није потребно да се мењају индекси осталих елемената, али код методе `shift()` морају да се помере индекси да су за 1 мањи да не би остале "рупе" у низу са недефинисаним вредностима.

```
var prijatelji = ["Miro", "Branka", "Ana"]; // dodela 3 stringa nizu
prijatelji.shift();   // izbacuje prvi element ("Miro") iz prijatelji
```

```
prijatelji = ["Miro", "Branka", "Ana"]; // ponovna dodela nizu
var x = prijatelji.shift(); // izbacena vrednost dodeljena x je "Miro"
// niz  prijatelji sada ima 2 elementa ["Branka", "Ana"]
```

Ако би се у једној команди (на пример у другој линији кода из претходног примера) применила метода (`prijatelji.shift();`) а затим у некој наредној линији кода (на пример у линији 4) извршила додела са `var x = prijatelji.shift();`, тада би се при додели избаченог елемента низа у ствари избацио и други елемент низа `prijatelji`. Сваки пут када се позове метода `shift()`, избаци се први елемент постојећег низа, без обзира да ли је урађена и додела избаченог елемента некој промењивој, па би тако двоструком применом методе `shift()`, била избачена два прва елемента низа на који се примењује ова метода.

Постоји још један начин да се обрише елемент низа. *JavaScript* низови су по типу објекти, па се елемент низа може обрисати коришћењем *JavaScript* оператора `delete`. Коришћење `delete` може да остави елемент који није дефинисан, тако да дужина низа остаје непромењена а податак је `undefined`. Због тога је боље да се користе методе `pop()` или `shift()` да не би остале "рупе" у низу.

Метода `unshift()` убацује нови елемент у низ као први елемент низа и помера (*unshifts*) све остале елементе у низу тако да је индекс сваког елемента низа за 1 већи у односу на индекс пре примене ове методе. Због убацивања новог елемента у низ, дужина низа се повећава за 1. На пример:

```
var prijatelji = ["Miro", "Branka", "Ana"]; // dodela 3 stringa nizu
prijatelji.unshift ("Ema");  //  dodaje novi element nizu ("Ema")
// niz  sada ima 4 elementa ["Ema", "Miro", "Branka", "Ana"]
```

Уколико се понови метода, биће додат нови елемент низу на првом месту и дужина ће бити повећана за још један.

Промена вредности елементу низа може да се уради преко индекса тог елемента. На пример:

```
var prijatelji = ["Miro", "Branka", "Ana"]; // dodela 3 stringa nizu
prijatelji[1] = "Ema"; // menja drugi element, indeks 1, u "Ema"
// elementi niza su ["Miro", "Ema", "Ana"]
```

Додавање новог елемента низа на крај може да се уради коришћењем дужине низа:

```
var prijatelji = ["Miro", "Branka", "Ana"]; // dodela 3 stringa nizu
prijatelji[prijatelji.length] = "Ema"; // dodaje novi element na kraj
// elementi niza duzine 4 su ["Miro", "Branka", "Ana", "Ema"]
```

За додавање више елемената низа може да се користи метода `splice()`. Метода `splice()` може да се искористи и за додавање и брисање елемената низа. На пример:

```
var prijatelji = ["Miro", "Branka", "Ana", "Ema"];
// originalni niz ["Miro", "Branka", "Ana", "Ema"]
prijatelji.splice(2, 0, "Jovan", "Pera"); // dodaje 2 elementa
// niz postaje ["Miro", "Branka", "Jovan", "Pera", "Ana", "Ema"]
```

Први параметар, 2, дефинише позицију где се додају (spliced in) нови елементи низа, а то је позиција на којој се у оригиналном низу налази елемент `Ana`. Други параметар, `0`,

дефинише колико елемената треба избацити из низа. У претходном примеру вредност је била 0, па зато није избачен ни један елемент низа. Затим следе параметри које треба убацити у низ, `"Jovan", "Pera"`, и број додатих елеменат у низ је онолики колико има параметара раздвојених зарезима.

Ако се некој променљивој додељује резултат методе `splice()`, та променљива ће садржати елементе низа који су обрисани.

Ако метода `splice()` има само прва два параметра, тада нема шта да се дода у низ. Уколико други параметар није 0, биће избачено из низа онолико елемената колико је наведено као други параметар. Ако је број другог параметра већи од броја елемената низа до краја низа, биће обрисани сви елементи низа до краја. Овом методом за брисање (избацивање) елемената низа не праве се "рупе" у низу са недефинисаним вредностима.

Метода `concat()` креира нови низ спајањем постојећих низова а што се назива конкатенација (*concatenation*, *join*). На пример:

```
var drugarice = ["Branka", "Ana", "Ema"];// dodela 3 stringa nizu
var drugovi = ["Miro"];                  // dodela 1 stringa nizu
var prijatelji = drugovi.concat(drugarice); // konkatenacija elementa
// novi niz postaje ["Miro", "Branka", "Ana", "Ema"]
```

Метода `concat()` не мења постојећи низ. Резултат конкатенације се додељује новом низу. Број параметара методе `concat()` може да буде већи од 1. Аргументи могу бити и стрингови а не само променљиве.

Метода `slice()` издваја део низа (*slice*) у нови низ. Резултат конкатенације се додељује новом низу, а број елемената оригиналног низа који ће бити у новом низу одређен је параметрима методе. На пример:

```
var prijatelji = ["Miro", "Branka", "Ana", "Ema"];
var drugarice = prijatelji.slice(2,4); // konkatenacija elementa
// novi niz postaje ["Ana", "Ema"]
```

Први параметар одређује индекс елемента од кога се издваја део за нови низ, на пример од индекса 2 који одговара трећем елементу оригиналног низа `"Ana"`. Други аргумент одређује индекс елемента оригиналног низа до кога се издвајају елементи, при чему се не укључује и тај елемент; индекс 4 је већи од највећег индекса оригиналног низа па ће у нови низ бити укључени сви елементи оригиналног низа. Ако се не специфицира вредност другог параметра, подразумева се да ће у новом низу бити сви елементи до краја оригиналног низа. Ако је број другог аргумента мањи или једнак првом аргументу, нови низ неће имати ни један елемент оригиналног низа.

Као и у случају примитивних вредности, *JavaScript* аутоматски конвертује низ у стринг при чему су елементи низа раздвојени зарезом. У примеру који следи се види да је вредност низа и аутоматска конверзија у стринг иста (`==` даје `true`), али да су различити само типови података (`===` даје `false`) зао што је низ типа објекат а није типа стринг:

```
var prijatelji = ["Miro", "Branka", "Ana", "Ema"];
(prijatelji == prijatelji.toString());  // vraca true
(prijatelji === prijatelji.toString()); // vraca false
```

3.2.1.2 Сортирање *JavaScript* низова

Метода `sort()` се користи за сортирање *JavaScript* низова. Низови се сортирају по азбучним реду. На пример:

```
var prijatelji = ["Miro", "branka", "Ana", "Ema", "@", "1", "2", "11"];
prijatelji.sort();  // sortiranje niza
// niz postaje ["1", "11", "2", "@", "Ana", "Ema", "Miro", "branka"]
```

Важно је уочити да се прво сортирају бројеви као стрингови па тек после тога стрингови који почињу словима. Број `"11"` је испред броја `"2"` зато што почиње са `1`. Прво се сортира по великим словима па тек после тога по малим словима, па је стога бб на крају сортиранох биза. Специјални карактери су између стрингова који почињу бројевима и стрингова који почињу словима, као на пример `"@"`.

Метода `reverse()` се користи за сортирање у обрнутом редоследу од постојећег. Ова метода не сортира по азбучном реду, већ последњи елемент постаје први, предпоследњи постаје други, ..., други елемент постаје предпоследњи, и први елемент постаје последњи, на пример:

```
var prijatelji = ["Miro", "branka", "Ana", "Ema", "@", "1", "2", "11"];
prijatelji.reverse();     // sortiranje obrnutim redosledom
// niz postaje ["11", "2", "1", "@", "Ema", "Ana", "branka", "Miro"]
```

Ако се два пута примени метода `reverse()`, добија се поново оригинални низ. Ако се употребе метода `sort()`, а затим и `reverse()`, добија се сортирани низ по обрнутом азбучном реду.

Подразумева се да метода `sort()`, сортира низове чији су елементи стрингови. Конверзијом бројева у стрингове, број `"11"` је мањи од броја `"2"` зато што почиње са `1`, а сортира се по првом слову (у овом случају по првом броју који је конвертован у стринг). Формално, ако се очекује да и бројеви буду сортирани по вредности а не по азбуци, потребно је користити додатну функцију, на пример:

```
var prijatelji = ["Miro", "branka", "1", "Ana", "Ema", "@", "2", "11"];
prijatelji.sort(function(a, b){return a - b});
// niz postaje ["2", "11", "@", "Ema", "Ana", "1", "branka", "Miro"]
```

Функција испитује узастопне вредности и сортира суседне бројеве као бројеве. Ако су групе бројева раздвојени стринговима, онда неће бити издвојени сви бројеви. Сортирање низова који имају и бројеве и струнгове неће дати очекивани резултат.

Ако се промени вредност која се враћа, на пример уместо `{return a - b}`, да буде `{return b - a}`, тада ће бројеви бити у опадајућем редоследу:

```
var prijatelji = ["Miro", "branka", "1", "Ana", "Ema", "@", "2", "11"];
prijatelji.sort(function(a, b){return b - a});
// niz postaje ["11", "2", "@", "Ema", "Ana", "1", "branka", "Miro"]
```

Могу се дефинисати и сложенији алгоритми сортирања, у зависности да ли оператор упоређивања даје негативне, нулте или потитивне вредност. Може се користити и сортирање по случајном узорку. На сајтовима о *JavaScript*-у могу се наћи бројни алгоритми за сортирање у складу са потребама апликације и брзине извршавања.

Након сортирања по растућем редоследу, први елемент низа садржи најмању вредност. Ако се сортира по опадајућем редоследу, први елемент има највећу вредност. На овај начин се могу наћи најмања и највећа вредност низа, обзиром да не постоји стандардна метода у *JavaScript*-у за одређивање најмање и највеће вредности низа. Наравно, овакав начин проналажења најмање и највеће вредности низа није ефикасан у случајевима низова велике дужине.

За проналажење најмање и највеће вредности низа који има елементе типа бројеви, могу се користити `Math.min()` и `Math.max()`. У случају да низ садржи елементе типа број и типа стринг, резултат може да буде и `NaN`. На сајтовима о *JavaScript*-у могу се наћи примери скрипт кодова за сортирање у оваквим случајевима (на пример на сајту https://www.w3schools.com/js/js_array_sort.asp).

Сортирање објеката који садрже вредности типа број, могу се урадити на сличан начин. Нека је дефинисан објекат:

```
var prijatelji = [
   {ime:"Ema", godina:2014},
   {ime:"Miro", godina:1957},
   {ime:"Ana", godina:2019}
];

prijatelji.sort(function(a, b){return a.godina - b.godina});
```

Сортирањем се добија

```
Miro 1957
Ema 2014
Ana 2019
```

У овом примеру се виде предности коришћења објеката уместо класичних низова. Упоређивање стрингова је нешто компликованије, на пример у зависности од тога да ли потребно да се велика слова конверују у мала.

3.2.1.3 Методе са итерацијама за *JavaScript* низове

Метода итерације функционишу за сваку ставку *JavaScript* низа.

Метода `forEach()` позива функцију (*callback, call-after* функција) по једном за сваки елемент низа. На пример:

```
var txt = "";
var prijatelji = ["Miro", "branka", 1, "Ana", "@", "2"];
prijatelji.forEach(myFunction);

function myFunction(value, index, array) {
  txt = txt + value + "<br>";
}
```

Извршавањем овог примера у прегледачу се добија:

```
Miro
branka
1
```

```
Ana
@
2
```

Функција генерише текст за приказ у *html* запису тако да се сваки елемент низа приказује у следећем реду у прегледачу. Како се у овом примеру користе три параметра, `value`, `index`, `array`, они могу да се изоставе и да се користи само први параметар. Метода ради тако што за сваки елемент низа, почев од првог до последњег, функција `myFunction` узима вредност низа и додаје га променљивој типа стринг `txt = txt + value + "<br>"`.

Ова метода не функционише у свим врстама прегледача.

Метода `map()` креира нови низ извршавајући функцију на сваком елементу низа. Ова метода не може да изврши функцију на елементу који нема вредност. На пример оператор за множење када је један операнд стринг произвест ће резултат `NaN`. Метода `map()` не мења оригинални низ. Следи пример у коме се сваки елемент низа множи са 2:

```
var numbers1 = [45, 4, 9 , "Ana", 16, 25];
// rezultat za numbers1 u pregledacu je 45,4,9,Ana,16,25
var numbers2 = numbers1.map(myFunction);
// rezultat za numbers2 u pregledacu je 90,8,18,NaN,32,50

function myFunction(value, index, array) {
  return value * 2;
}
```

И у овом примеру могу да се користе три параметра, `value`, `index`, `array`, али и само први параметар, `value`. Ова метода не функционише у свим врстама прегледача.

Метода `filter()` креира нови низ чији елементи задовољавају задати услов, на пример:

```
var numbers = [45, 4, 9 , "Ana", 16, 25];
// rezultat za numbers u pregledacu je 45,4,9,Ana,16,25
var veciOd18 = numbers.filter(myFunction);
// rezultat za veciOd18 u pregledacu je 45,25

function myFunction(value, index, array) {
  return value > 18;
}
```

У примеру је показано како је генерисан нови низ који садржи бројеве веће од 18. Елементи који нису типа број неће се појавити у новом низу. И у овом примеру могу да се користе три параметра, `value`, `index`, `array`, али и само први параметар, `value`. Ова метода не функционише у свим врстама прегледача.

Метода `reduce()` извршава функцију на сваком елементу низа и враћа једну вредност. Метода се извршава са лева у десно по елементима низа. Ова метода не мења оригинални низ. У следећем примеру је пример да израчуна збир свих елемената низа:

```
var numbers1 = [45, 4, 9, "10", 16, 25];
// rezultat za numbers1 u pregledacu je 45,4,9,10,16,25
var zbir = numbers1.reduce(myFunction);
```

```
// rezultat za zbir u pregledacu je 58101625

function myFunction(total, value, index, array) {
  return total + value;
}
```

Резултат који се добија у прегледачу није збир свих бројева зато што нису сви елементи низа типа број:

```
58101625
```

Функција сабира редом од првог броја у низу, а то је `58` као збир бројева `45`, `4`, `9`. Четврти елемент је стринг `"10"`, па функција конверује израчунат збир `58` у стринг `"58"`, и све остале елементе додаје (извршава конкатенацију) као стрингове у коначан резултат `58101625`. У случају да је уместо `"10"` у низу `"NaN"`, резултат ће бити `58NaN1625`. Међутим, у случају да је уместо `"10"` у низу `NaN`, без наводника, резултат ће бити `NaN`, зато што свака операција са `NaN` даје резулта `NaN`, па је и коначна вредност израчунавања `NaN`.

И у овом примеру могу да се користе четири параметра, `total`, `value`, `index`, `array`, али и само прва два параметра, `total` и `value`. Ова метода не функционише у свим врстама прегледача.

Метода `reduceRight()` извршава функцију на исти начин и по истим правилима као метода `reduce()`, с тим што елементе низа узима од последњег ка првом (или са десна у лево).

Метода `every()` проверава да ли сви елементи низа задовољавају задати услов. Следећи пример показује да ли су сви елементи низа имају вредност већу од броја 18:

```
var numbers = [45, 40, 90, "100", 160, 25];
var allOver18 = numbers.every(myFunction); // vraca true

function myFunction(value, index, array) {
  return value > 18;
}
```

Иако је четврти елемент стринг `"100"`, он се најпре конвертује у тип број, и зато је резултат `true`. Када би било који елемент имао вредност мању од специфициране, или био типа стринг а да није састављен од цифара, добио би се резултат `false`. И за ову методу могу да се користе три параметра, `value`, `index`, `array`, али и само први параметар, `value`. Ова метода не функционише у свим врстама прегледача.

Метода `some()` проверава да ли бар један елемент низа задовољавају задати услов. Следећи пример показује да неки елементи низа имају вредност већу од броја 18:

```
var numbers = [4.5, 4, 9, 16, 25];
var someOver18 = numbers.some(myFunction); // vraca true

function myFunction(value, index, array) {
  return value > 18;
}
```

Ова метода враћа резултат `false` само ако сви елементи низа не задовољавају задати услов. Ова метода не функционише у свим врстама прегледача.

Метода `indexOf()` претражује да ли низ садржи неки елемент и враћа позицију првог појављивања елемента у низу. Позиција првог елемента је 0, а последњег елемента за 1 мања од дужине низа, на пример:

```
var prijatelji = ["Ana", "Miro", "Ana", "Ema","Ana"];
var ana = prijatelji.indexOf("Ana");  // vraca 0
```

Ова метода може да има два параметра, при чему други параметар означава од које позиције се претражује. На пример, да је коришћено `.indexOf("Ana", 3)`, добијена би позиција трећег појављивања стринга `"Ana"` који је 4. Уколико није пронађен тражени стринг у низу, добија се резултат -1. Ако је други параметар негативан, претраживање почиње од краја низа померено за вредност другог атрибута, па до последњег елемента низа.

Метода `lastIndexOf()` ради исто као и `indexOf()` с тим да проналази последње појављивање елемента у низу. За исти пример се добија да метода `lastIndexOf()` враћа последње појављивања стринга `"Ana"` на позицији 4. Метода `lastIndexOf()` може да има два параметра, при чему други параметар има исто значење као и у `indexOf()`. Методе `indexOf()` и `lastIndexOf()` не функционише у свим врстама прегледача.

Метода `find()` враћа вредност првог елемента низа који задовољава задати услов, на пример:

```
var numbers = [4, 9, "100", 25, 29];
var first = numbers.find(myFunction); // vraca 100

function myFunction(value, index, array) {
  return value > 18;
}
```

Иако је трећи елемент типа стринг, он се састоји од цифара, па ће бити конвертован у број пре тестирања да ли је испуњен услов.

И у овом примеру могу да се користе три параметра, `value`, `index`, `array`, али и само први параметар, `value`. Ова метода не функционише у свим врстама прегледача.

Метода `findIndex()` враћа позицију (индекс) првог елемента низа који задовољава задати услов, на пример:

```
var numbers = [4, 9, "100", 25, 29];
var first = numbers.findIndex(myFunction); // vraca 2

function myFunction(value, index, array) {
  return value > 18;
}
```

Ова метода ради на исти начин као `find()`, с тим да уместо вредности елемента низа враћа позицију појављивања елемента низа.

3.2.2 *JavaScript* објекти за датум и време

JavaScript објекти за време (и датум) називају се *JavaScript Date Objects*. Важно је да се зна који је тренутни датум и време или да се израчуна период од неког тренутка до другог. На пример следећи запис показује време писања овог поглавља

```
Sat Aug 15 2020 08:56:35 GMT+0200 (Central European Summer Time)
```

Овакав запис није у форми од највеће вредности времена ка најмањој, али је прилагођен потребама коришћења; на пример прво је приказан дан у недељи, па месец, затим дан, и тек онда година после који следи време. Да би време било приказано по вредности, природније би било за рачунар да је записано у форми:

```
Година: 2020
Месец: 08
Дан: 15
Час: 08
Минути: 56
Секунде: 35
```

Затим иду додатна објашњења у вези времена:

```
Дан у недељи: Sat
Временска зона GMT+0200 (Central European Summer Time)
```

Ако би се овај запис користио за потребе означавања броја по вредности који не би доводио до забуне у интерпретацији, на пример за означавање тренутка креирања једног фајла, и да истовремено буде разумљив кориснику, тада би запис био у форми 4 броја за годину `2020`, два броја за месец `08`, два броја за дан `15`, и по два броја за час`08`, минуте `56` и секунде `35`., односно као један број `20200815085635`. За различите временске зоне овака запис би могао да узрокује грешку. На пример ако пратите време испоруке поштанске пошиљке која је послата из Србије у земљу која је у другој временској зони, може да се појави разлика од 2 сата за исто време које се приказује у једној или другој држави. Да се не би компликовао запис, ако се подразумева да је иста временска зона, тада се коментар за временску зону посебно записује, и тиме поједностављује запис времена. Због тога се време рачуна према временској зони прегледача.

Постоје 4 начина за креирање објекта за време:

```
new Date()
new Date(година, месец, дан, час, минуте, секунде, милисекунде)
new Date(милисекунде)
new Date(стринг типа време)
```

На пример, нови *date* објекат се може креирати на следећи начин:

```
var d1 = new Date(); // u vreme pisanja primera  Sat Aug 15 2020 08:56:35
var d2 = new Date(2020, 7, 15, 8, 56, 35, 0); // Sat Aug 15 2020 08:56:35
var d3 = new Date(2020, 7, 15, 8, 56, 35);    // Sat Aug 15 2020 08:56:35
var d4 = new Date(2020, 7, 15, 8, 56);        // Sat Aug 15 2020 08:56:00
var d5 = new Date(2020, 7, 15, 8);            // Sat Aug 15 2020 08:00:00
var d6 = new Date(2020, 7, 15);               // Sat Aug 15 2020 00:00:00
var d7 = new Date(2020, 7);                   // Sat Aug 01 2020 00:00:00
var d8 = new Date(2020);                      // Thu Jan 01 1970 01:00:02
```

Date објекти су типа *static*. Рачунарско време се мења (откуцава), али *date objects* се не мења. JavaScript рачуна месеце од 0 до 11. Јануар је 0, август је 7, а децембар је 11. Ако се изостави месец, и постоји само један аргумент, то није година већ милисекунде од 1970. године:

```
var d9 = new Date(2020);         // Thu Jan 01 1970 01:00:02
```

За означавање времена пре 2000-те године, може се користити и број од једне или две цифре за годину. На пример:

```
var d10 = new Date(99, 11, 4);  // Sat Dec 04 1999 00:00:00
var d11 = new Date(9, 12, 14);  // Fri Jan 14 1910 00:00:00
var d12 = new Date(9, 7, 4);    // Wed Aug 04 1909 00:00:00
```

У другом реду је коришћен број 12 за месец; како је број за децембар 11. То онда значи да се година повећа за 1, а месец се умањује за 12, па постаје 0, односно јануар.

Резултат који приказује прегледач је стринг, на пример за `d2`. Део стринга може да се користи као аргумент да би се креирао нови *date* објекат у комплетном запису:

```
var d13 = new Date("Sat Aug 15 2020 08:56:35");
// vraca Sat Aug 15 2020 08:56:35 GMT+0200 (Central European Summer Time)
```

Формат приказивања времена може да се разликује од формата у коме се пдаци памте. Формат за *date* објекат је такав да га човек одмах разуме, али то није формат који се памти у рачунару. Као што се бројеви приказују декадном систему, а пампе у бинарном систему, тако се и подаци за време приказују у формату разумљивом кориснику да одма препозна годину или дан или сат и минитр, а у меморији се подаци чувају као цели бројеви исказани у милисекундама (`ms`). На пример:

```
var d14 = new Date("Aug 15 2020 08:56:35"); // Aug 15 2020 08:56:35
var d15 = 1*d14;  // vraca 1597474595000 u ms
```

Променљива `d14` се приказује у формату за време, `Sat Aug 15 2020 08:56:35 GMT+0200 (Central European Summer Time)`, а променљива `d15` се приказује као цео број `1597474595000` у ms. У меморији рачунара се складишти цео број у ms као податак за време. У овом случају смо искористили аутоматску коверзију са оператором за множење. Множење са 1 неће променити вредност операнда који је за време, али ће га конвертовати у цео број и приказати као тип податка број, а не као податак за време.

Сада можемо да проверимо и које време одговара вредности 0: то је 1. јануар 1970 године у 0 сати 0 минута и 0 секунди по *Universal Time* (*UTC*); међутим за рачунање у временској зони у Србији то је 1. јануар 1970 године у 1 сат 0 минута и 0 секунди по GMT+0100 (*Central European Standard Time*):

```
var d16 = new Date("Jan 1 1970 1:00:00"); // Jan 01 1970 00:00:00 UTC
var d17 = 1*d16;  // vraca 0 u ms za GMT+0100
```

Негативан број у ms показује време пре 1. јануара 1970 године у 0 сати 0 минута и 0 секунди по UTC, а позитиван број показује време после 1. јануара 1970 године у 0 сати 0 минута и 0 секунди по UTC. Као илустрација, један дан је 24 сати, а то је 86 400 000 ms.

За креиране *date* објекте могу се користити бројни методи да би се радило са њима. Методе *date* омогућавају да се добије или постави *date* објекат као што су година, месец, дан, час, минута, секунда и милисекунда коришћењем локалног времена или UTC или

GMT времена. *GMT* (*Greenwich Mean Time*) је временска зона, а *UTC* (*Coordinated Universal Time*) је стандард за време.

Подразумева се да *JavaScript* приказује датум и време у развијеном текстуалном облику:

```
var d = new Date("Sat Aug 15 2020 08:56:35");
// vraca Sat Aug 15 2020 08:56:35 GMT+0200 (Central European Summer Time)
```

Када се датум и време приказују из *html* документа у прегледачу, аутоматски се конвертује у стринг коришћењем `toString()` методе, односно као да је запис: `d.toString()`, `Sat Aug 15 2020 08:56:35 GMT+0200 (Central European Summer Time)`.

Може да се користи и метода `toUTCString()` која конвертује *date* у UTC стринг (стандард за приказ датума и времена), који је за овај пример у `Sat, 15 Aug 2020 06:56:35 GMT`.

Метода `toDateString()` конвертује датум у читљивију форму, `Sat Aug 15 2020`.

Метода `toISOString()` конвертује датум у стринг коришћењем *ISO* стандардног формата: `2020-08-15T06:56:35.000Z`

3.2.3 *JavaScript* формати за датум

Генрално, постоје три формата за *JavaScript* датуме:

Тип	Пример
ISO Date	`"2015-03-25"` (*The International Standard*)
Short Date	`"03/25/2015"`
Long Date	`"Mar 25 2015"` или `"25 Mar 2015"`

JavaScript ISO date формати, као улазни подаци, стрикно следе *ISO* стандард. Остали формати могу да буду и другачије приказани у различитим прегледачима.

JavaScript ISO date формати, као излазни подаци, приказују у прегледачима цео текст као стринг. ISO 8601 је међународни стандард за приказ датума и времена. Синтакса ISO 8601 стандарда је YYYY-MM-DD где је година приказана са 4 цифре YYYY, месец са две цифре MM, а дан такође са две цифре DD, на пример:

```
var d = new Date("2020-08-15");
// Sat Aug 15 2020 02:00:00 GMT+0200 (Central European Summer Time)
```

Важно је уочити да се као улазни податак у стринг формату месеци означавају бројем у години, на пример осми месец август је означен бројем 8, а не бројем 7, како се означава у формату са бројевима, нпр. `new Date(2020, 7, 15);`. Због постојања временских зона, приказ датума се може разликовати за један дан. Подаци о времену су постављени на 0, тако да су минуте и секунде `:00:00`, али сати нису 0 већ су постављени на вредност коју одређује временска зона, на пример на `02:` због `GMT+0200`.

ISO формат за датум може да се означи само са подацима за годину и месец YYYY-MM, на пример:

```
var d = new Date("2020-08");
// Sat Aug 01 2020 02:00:00 GMT+0200 (Central European Summer Time)
```

Дан у години се поставља на 1, на пример на `Aug 01`. Због временских зона, може доћи до разлике за један дан, на пример да буде `February 28` или `March 01`.

ISO формат за датум може да се означи само са подацима за годину YYYY, на пример:

```
var d = new Date("2020");
// Wed Jan 01 2020 01:00:00 GMT+0100 (Central European Standard Time)
```

Дан у години се поставља на 1, а месец на јануар, на пример на `Jan 01`. Због временских зона, може да се разликује приказ 31 децембра 31 и 1 јануара. У овом примеру су сати постављени на `01`, зато што је `GMT+0100`.

ISO формат за датум може да садржи и податке о времену у формату година-месец-данTсати:минуте:секундеZ, на пример, YYYY-MM-DDTHH:MM:SSZ):

```
var d = new Date("2020-08-15T08:56:35Z");
// Sat Aug 15 2020 10:56:35 GMT+0200 (Central European Summer Time)
```

Као улазни податак у стринг формату месеци означавају редним бројем у години, на пример осми месец август је означен бројем 8, а не бројем 7 као у формату са бројевима. У овом примеру су сати постављени на `08`, а приказују се као `10` зато што је `GMT+0200`. Слово `T` служи да раздвоји датум од времена, а слово `Z` служи да означи крај записа.

Ако се жели другачије време у односу на UTC потребно је да се уместо слова Z дода са знаком плус или минус +HH:MM или -HH:MM, на пример:

```
var d = new Date("2020-08-15T08:56:35-06:03");
// Sat Aug 15 2020 16:59:35 GMT+0200 (Central European Summer Time)
```

Приказано време ће бити веће од постављеног за 2 због `GMT+0200`, и додатно за 6 сати и 3 минута због `-06:03`, тако да је приказано време `16:59:35`.

Изостављање слова `T` или `Z` у стрингу за датум-време може да врати различите резултате у различитим прегледачима

Када се поставља датум (*setting a date*) без специфицирања временске зоне (*time zone*), *JavaScript* ће користити временску зону прегледача. Када се узима датум без специфицирања временске зоне, резултат се конвертује у временску зону прегледача. Другим речима, ако се датум-време креира у *GMT* (*Greenwich Mean Time*), запис датум-време ће бити конвертован у *CDT* (*Central US Daylight Time*) ако корисник претражује из Америке.

Скраћени стринг запис за датум је у формату "MM/DD/YYYY", на пример:

```
var d = new Date("08/15/2020");
// Sat Aug 15 2020 00:00:00 GMT+0200 (Central European Summer Time)
```

Важно је уочити да у Србији користимо запис дан-месец-година, што може да проузрокује неразумевање записа ако и месец и дан имају вредност у опсегу од 1 до 12, па по основу тога да се грешком замене позиције за дан и месец.

У неким прегледачима може доћи до грешке ако стринг запис нема водеће нуле за месец и дан, на пример ако су месец и дан у опсегу од 1 до 12:

```
var d = new Date("8/5/2020"); // nema gresku
// Wed Aug 05 2020 00:00:00 GMT+0200 (Central European Summer Time)
```

У неким прегледачима може да ради, у некима ће јавити грешку `Invalid Date`, или ће вратити резултат `NaN`.

```
var d = new Date("8-5-2020"); // ima gresku, Invalid Date ili NaN
var d = new Date("2020-8-5"); // nema gresku
// Wed Aug 05 2020 02:00:00 GMT+0200 (Central European Summer Time)
```

Дужи запис за датум (*JavaScript Long Dates*) је обично у формату "MMM DD YYYY", при чему је месец дат описно са прва три слова месеца а не бројем, на пример:

```
var d = new Date("Mar 15 2020");       // Mar 15 2020
var d = new Date("15 Mar 2020");       // Mar 15 2020
var d = new Date("January 15 2020");   // Jan 15 2020
var d = new Date("Jan 15 2020");       // Jan 15 2020
var d = new Date("JANUARY, 15, 2020"); // Jan 15 2020
```

Обзиром да је месец задат словима а дан бројем, сада не може да дође до грешке и могући су сви облици записа као улазних података за датум, у форми месец дан година `"Mar 15 2020"`, дан месец година `"15 Mar 2020"`, месец дан година `"January 15 2020"`, месец дан година `"JANUARY, 15, 2020"`, са бланко раздвајањем или са раздвајањем зарезима, са свим великим словима за месец или са првим великим словом за месец. У српском језику се месеци пишу малим почетним словом, док се у енглеском користи прво велико слово за назив месеца.

Пун назив месеци у *JavaScript*-у су `"January"`, `"February"`, `"March"`, `"April"`, `"May"`, `"June"`, `"July"`, `"August"`, `"September"`, `"October"`, `"November"`, `"December"`.

Пун назив дана у недељи у *JavaScript*-у су `"Sunday"`, `"Monday"`, `"Tuesday"`, `"Wednesday"`, `"Thursday"`, `"Friday"`, `"Saturday"`;

Метода parse() може да се искористи да се датум-време запис конвертује у целобројну вредност у циљу издвајања информације о датуму или времену, (*parsing dates*) . На пример, додели се променљивој `msec` типа број вредност изражена у `ms` почев од 1.1.1970:

```
var msec = Date.parse("August 15, 2020");  // vraca 1597442400000 u ms
```

Овако добијена бројна вредност сада може да се користи за приказ датума и времена:

```
var msec = Date.parse("August 15, 2020");// vraca 1597442400000 u ms
var d = new Date(msec);
// Sat Aug 15 2020 00:00:00 GMT+0200 (Central European Summer Time)
```

На овај начин се постиже конверзија датума у број и кнверзија броја у датум.

3.2.4 *JavaScript Get Date* методе

Методе наведене у табели могу да се користе за добијање информација из *date* објеката.

Метода	Опис
`getFullYear()`	Добијање године (***year***) као четвороцифрени број (yyyy)
`getMonth()`	Добијање месеца (***month***) као број од 0 до 11
`getDate()`	Добијање дана (***day***) као број од 1 до 31
`getHours()`	Добијање сата (***hour***) као број од 0 до 23
`getMinutes()`	Добијање минуте (***minute***) као број од 0 до 59
`getSeconds()`	Добијање секунди (***second***) као број од 0 до 59
`getMilliseconds()`	Добијање милисекунди, ms, (***millisecond***) као број од 0 до 999
`getTime()`	Добијање времена (*time milliseconds*) у односу на 1.1.1970.

`getDay()`	Добијање дана у недељи (***weekday***) као број од 0 до 6
`Date.now()`	Добијање времена (*time*) према ECMAScript 5

Следе примери за методе из табеле:

```
var d = new Date("2020-08-15T08:56:35Z");// Aug 15 2020 10:56:35 GMT+0200
d.getFullYear();     // vraca 2020
d.getMonth();        // vraca 7 (08 - 1)
d.getDate();         // vraca 15
d.getHours();        // vraca 10 (T08 + GMT+0200)
d.getMinutes();      // vraca 56
d.getSeconds();      // vraca 35
d.getMilliseconds(); // vraca 0
d.getTime();         // vraca 1597481795000
d.getDay();          // vraca 6 (Sat, subota)
var n = Date.now();  // vraca 1597608908956
// Sun Aug 16 2020 22:15:08 GMT+0200 (Central European Summer Time)
```

Први дан у недељи са позицијом 0 је недеља, а последњи дан у недељи је субота са позицијом 6. Пун назив месеца и дана може да се додели низу, и тада се користи вредност коју враћа метода за датум (`getMonth()`, `getDay()`) као индекс низа да се добије назив месеца или дана као стринг. Низ не мора да садржи енглеске називе за месеце или дане, већ се у низу могу налазити српски називи.

Методе наведене у наредној табели могу да се користе за добијање информација из *date* објеката као подаци за *UTC* (*Universal Time Zone dates*). Важе иста правила као у претходној табелу, с том разликом да је за *UTC*.

Метода	**Опис за UTC**
`getUTCFullYear()`	Добијање године (***year***) као четвороцифрени број
`getUTCMilliseconds()`	Добијање милисекунди, ms, као број од 0 до 999
`getUTCMinutes()`	Добијање минуте (***minute***) као број од 0 до 59
`getUTCSeconds()`	Добијање секунди (***second***) као број од 0 до 59
`getUTCDate()`	Добијање дана (***day***) као број од 1 до 31
`getUTCHours()`	Добијање сата (***hour***) као број од 0 до 23
`getUTCMonth()`	Добијање месеца (***month***) као број од 0 до 11
`getUTCDay()`	Добијање дана у недељи (***weekday***) као број од 0 до 6

3.2.5 *JavaScript Set Date* методе

Методе постављања датума у *JavaScript*-у (*set date*), омогућавају да се поставе вредности за године (*years*), месеце (*months*), дане (*days*), часове (*hours*), минуте (*minutes*), секунде (*seconds*), милисекунде (*milliseconds*) за објекте типа *date*.

Методе наведене у табели се користе за постављање дела датума и времена.

Метода	**Опис**
`setDate()`	Поставља дан као број од 1 до 31
`setFullYear()`	Поставља годину као број од (опционо месец и дан)
`setHours()`	Поставља час као број од 0 до 23
`setMilliseconds()`	Поставља милисекунде као број од 0 до 999

`setMinutes()`	Поставља минуте као број од 0 до 59
`setMonth()`	Поставља месец као број од 0 до 11
`setSeconds()`	Поставља секунде као број од 0 до 59
`setTime()`	Поставља време као број од 1.1.1970 у милисекундама

Следе примери за постављање датума и времена

```
var d = new Date("2020-08-15T08:56:35Z"); // Aug 15 2020 10:56:35 GMT+0200
```

а затим извршавања само једног од наредних сетовања

```
d.setFullYear(2020);                      // Aug 15 2021 10:56:35 GMT+0200
d.setFullYear(2020, 10, 17);              // Nov 17 2020 10:56:35 GMT+0100
d.setMonth(11);                           // Dec 15 2020 10:56:35 GMT+0100
d.setDate(23);                            // Aug 23 2020 10:56:35 GMT+0200
d.setDate(d.getDate() + 50);              // Oct 04 2020 10:56:35 GMT+0200
d.setHours(22);                           // Aug 15 2020 22:56:35 GMT+0200
d.setMinutes(30);                         // Aug 15 2020 10:30:35 GMT+0200
d.setSeconds(30);                         // Aug 15 2020 10:56:30 GMT+0200
```

Методе дозвољавају да се преузме вредност и дода број, као на пример што је урађено за дан, па је због броја, који је већи од 31 који одговара дану, повећање довело и до промене месеца од августа у октобар.

Датуми и време се могу веома једноставно користити за компарацију и аритметичка израчунавања. На пример, нека су позната два датума са временима; треба израчунати разлику у милисекундама а затим одредити број дана између ова два тренутка:

```
var d1, d2, d3;                         // deklarisanje promenljivih
d1 = new Date("2020-08-15T08:56:35Z"); // Aug 15 2020 10:56:35 GMT+0200
d2 = new Date("2022-05-29T00:00:00Z"); // May 29 2022 02:00:00 GMT+0200
d3 = ((d2.getTime()-d1.getTime())/1000/60/60/24).toFixed(0); // broj dana
```

Најпре се све три променљиве декларишу, `d1`, `d2`, `d3`. Променљива `d1` се постави на један датум са специфікованим и временом, друга се постави на други датум `d2`. Обе вредности добијају вреност целог броја у милисекундама. Разлика ове две вредности је `d3`, број између ова два тренутка изражен у милисекундама. Да би се добила вредност у секундама потребно је да се број подели са `1000`. Број у секундама треба да се подели са `60` да би се добила вредност у минутама, а затим да се подели са још `60`, да би се добила вредност у сатима. Вредност у сатима треба поделити са `24`, и тада се добија вредност у данима. Вредност у данима је рационални број који има и разломљени део; зато је искоришћена метода `toFixed(0)`, да би се добио цео број као приближна вредност дана између ова два временска тренутка. Временски опсег који се рачуна је већи од једне године, и ручно израчунавање би било компликовано јер је потребно урачунати и број дана у години који може да буде 365 или 366 због преступне године. Коришћење објеката за датум и време значајно олакшава израчунавање и избегавање грешке.

На исти начин се могу користити објекти *date* за компарацију два датума који су задати у различитом формату, или за гранање програма у зависности да ли је вредност разлике мања или већа од нуле.

3.2.6 Питања и задаци за вежбу

1. Написати пример декларисања *JavaScript* низа.
2. Објаснити разлоге за прављење променљиве као *JavaScript* низ.
3. Објаснити разлику између низова и објеката у *JavaScript*-у.
4. Како може да се одреди да ли је променљива низ или објекат у *JavaScript*-у?
5. Како се приступа елементу *JavaScript* низа?
6. Како се добија дужина *JavaScript* низа?
7. Објаснити ратлику између метода `push()`, `pop()` и `shift()` за *JavaScript* низове?
8. Како се убацују нови елементи у *JavaScript* низ?
9. Шта је конкатенација у *JavaScript* низу?
10. Како се користи *callback* функција код *JavaScript* низова?
11. Када се појављују "рупе" у *JavaScript* низу?
12. Написати пример за сортирање *JavaScript* низа.
13. Написати пример за *JavaScript* објекат за датум.
14. У којим јединицама се приказују датум када се на пример добије 1597474595000?
15. Објаснити шта ради *JavaScript* Get Date метода.
16. Објаснити шта ради *JavaScript* Set Date метода.

3.3 Математички *JavaScript* објекти

Резиме

Циљ овог поглавља је да студент научи да ради са математичким *JavaScript* објектима и са *JavaScript* случајним бројевима.

Увод у математичке JavaScript објекте

JavaScript математички објекат (*Math object*) омогућава да се извршавају задаци на бројевима. Математички објекти могу да буду математичке константе као што су:

```
Math.PI;      // vraca konstantu pi, 3.141592653589793
Math.E        // vraca Ojlerovu konstantu E, 2.718281828459045
Math.SQRT2    // vraca kvadratni koren iz 2, 1.4142135623730951
Math.SQRT1_2  // vraca kvadratni koren iz 1/2, 0.7071067811865476
Math.LN2      // vraca prirodni logaritam od 2, 0.6931471805599453
Math.LN10     // vraca prirodni logaritam od 10, 2.302585092994046
Math.LOG2E    // vraca logaritam od E za osnovu 2, 1.4426950408889634
Math.LOG10E   // vraca logaritam od E za osnovu 10, 0.4342944819032518
```

Математички објекти могу да немају ни један аргумент, на пример случајну вредност:

```
Math.random();     // vraca slucajan broj iz opsega od 0 do 1
// npr 0.555320333819406, 0 je ukljucena u opseg, a 1 nije ukljucena
```

Математички објекти могу да имају један аргумент, на пример:

```
Math.round(4.7);    // vraca zaokruzenu vrednost na blizi broj, 5
Math.round(4.4);    // vraca zaokruzenu vrednost na blizi ceo broj, 4
Math.sqrt(64);      // vraca kvadratni koren broja, 8
Math.abs(-4.3);     // vraca absolutnu vrednost, 4.3
```

```
Math.ceil(4.4);     // vraca zaokruzenu vrednost na veci ceo broj, 5
Math.floor(4.7);    // vraca zaokruzenu vrednost na manji ceo broj, 4
```

Математички објекти могу да имају један аргумент у неким јединицама које се подразумевају, на пример у радијанима. Ако је вредност аргумента у другим јединицама, потебно је да се уради конверзија аргумента у радијане, на пример:

```
Math.sin(90 * Math.PI / 180);   // vraca 1 (sinus od 90 stepeni)
Math.cos(0 * Math.PI / 180);    // vraca 1 (kosinus od 0 stepeni)
```

Математички објекти могу да имају два аргумента, на пример:

```
Math.pow(8, 2);      // vraca 8 na drugi stepen, 64
```

Математички објекти могу да имају листу аргумента, на пример:

```
Math.min(0, 149, 45, 20, -7, -210); // vraca najmanju vrednost iz liste, -210
Math.max(0, 149, 45, 20, -7, -210); // vraca najvecu vrednost iz liste, 149
```

За разлику од дригих глобалних објеката, математички објекти немају конструктор. Методе и особине су статичке. Све методе и особине (константе) могу да се користе и без претходног креирања математичког објекта.

У следећој табели су дате методе математичких објеката по абецедном реду.

Метода	Опис
abs(x)	Враћа апсолутну вредност од x, |x|
acos(x)	Враћа аркус косинус од x, у радијанима
acosh(x)	Враћа хиперболични аркус косинус од x
asin(x)	Враћа аркус синус од x, у радијанима
asinh(x)	Враћа хиперболични аркус синус од x
atan(x)	Враћа акустангес од x као број између -π/2 и π/2 радијана
atan2(y, x)	Враћа акустангес y и x компоненте
atanh(x)	Враћа the хиперболични аркустангес од x
cbrt(x)	Враћа трећи корен од x
ceil(x)	Враћа x, заокружена на већи цео број
cos(x)	Враћа косинус од x (x у радијанима)
cosh(x)	Враћа хиперболични косинус од x
exp(x)	Враћа E на степен од x, E^x, (E је 2.718281828459045)
floor(x)	Враћа x, заокружена на мањи цео број
log(x)	Враћа природни логаритам од x, за базу E
max(x, y, z, ..., n)	Враћа највећу вредност листе бројева
min(x, y, z, ..., n)	Враћа најмању вредност листе бројева
pow(x, y)	Враћа вредност x на степен y, x^y
random()	Враћа случајан број између 0 и 1, 1 није укључен као резултат
round(x)	Враћа x најближи цео број за x
sin(x)	Враћа синус од x (x у радијанима)
sinh(x)	Враћа хиперболични синус од x
sqrt(x)	Враћа квадратни корен од x, за√(x)
tan(x)	Враћа тангес угла x
tanh(x)	Враћа хиперболични тангес x
trunc(x)	Враћа цео део броја x

3.3.1 *JavaScript* случајни бројеви

JavaScript враћа број који има случајну вредност између 0 (укључујући 0) и 1 (искључујући 1) коришћењем `Math.random()`.

Генератор случајних бројева је погодан за симулацију случајних процеса. На пример, ако се прави апликација у којој се појављује бацање коцкице, која може да има случајну вредност од 1 до 6, тада се случајно генерисан број са `Math.random()` помножи са 6, а затим се са `1+Math.trunc(6*Math.random())`, добије случајно појављивање целог броја из опсега од 1 до 6. Наравно, може се користити и `floor` уместо `trunc`.

Ако се програмира случајна функција која треба да укључује и најмању и највећу вредност, тада би могао да се користи следећи код:

```
function getRndInteger(min, max) {
  return Math.floor(Math.random() * (max - min + 1) ) + min;
}
```

Генерисање *JavaScript* случајних бројева је посебно корисно за тестирање исправности програмског кода када се случајно задају улазне вредности и великим бројем понављања извршавања кода насумично проналази грешка ако израчунавање није добро програмирано.

3.3.2 Питања и задаци за вежбу

1. Колико аргумената могу да имају *JavaScript* математички објекти?
2. Написати пример за израчунавање апсолутне вредности у *JavaScript*-у.
3. Објаснити како се добијају случајне вредности у *JavaScript*-у.

3.4 Логичке *JavaScript* променљиве

Резиме

Циљ овог поглавља је да студент детаљно савлада рад са логичким *JavaScript* подацима као кључним подацима за рад са петљама и условним операторима. Посебно студент треба да разуме примену логичких података за потребе *JavaScript* компарације, условне наредбе и *Switch* наредбе.

Увод у логичке JavaScript променљиве

JavaScript логичка вредност (*Boolean*) може да има само две вредности `true` и `false`. У програмирању је често потребно да постоје променљиве које могу да имају само две вредности као на пример Да/Не (`YES / NO`), Укључено/Искључено (`ON / OFF`), тачно/нетачно (`TRUE / FALSE`) или 1 / 0 (овде коса црта не значи 1 подељено са 0, већ је сепаратор две вредности). За овакав тип података се каже да су логичке или *Boolean data type*.

Може да се користи функција Boolean() да ли је неки израз тачан или нека логичка променљива има логичку вреднос, на пример:

```
Boolean(10 > 9)       // vraca true
```

```
(10 > 9)                // vraca true
10 > 9                  // vraca true
```

Условни оператори раде компарацију две вредности и у зависности од услова (да ли су две вредности исте (==), прва вредност је већа од друге (>), или је прва вредност мања од друге (<), враћају једну вредност `true` или `false`.

Све вредности као аргумент од `Boolean()`, вратиће резултат `true`:

```
var b1 = Boolean(100);              // vraca true
var b2 = Boolean(7.3);              // vraca true
var b3 = Boolean(-14);              // vraca true
var b4 = Boolean(10 / 0);           // vraca true iako je 10 / 0 = Infinity
var b5 = Boolean("Miro");           // vraca true
var b6 = Boolean("false");          // vraca true
var b7 = Boolean(7.3 + 1 + 1.17);   // vraca true
```

Међутим, када је вредност 0, или -0, или празан стринг, добија се резултат `false`:

```
var b8 = Boolean(0);                // vraca false
var b9 = Boolean(-0);               // vraca false
var b10 = Boolean("");              // vraca false
```

Резултат `false` се добија када је вредност 0, или -0, празан стринг или `NaN`:

```
var x1;                     // nije dodeljena vrednost, undefined
Boolean(x1);                // vraca false
var x2 = null;              // dodeljena vrednost null
Boolean(x2);                // vraca false
var x3 = false;             // dodeljena vrednost false
Boolean(x3);                // vraca false
var x4 = 10 / "H";          // dodeljena vrednost NaN
Boolean(x4);                // vraca false
```

Логичке променљиве могу да буду објекти иако су *JavaScript booleans* примитивне вредности креирани из константе. Логичке променљиве могу да се дефинишу као објекти коришћењем кључне речи `new`. На пример:

```
var x = false;              // typeof x vraca boolean
var y = new Boolean(false); // typeof y vraca object
var z = new Boolean(false); // typeof z vraca object
// (x == y) je true zato sto x i y imaju iste vrednosti
// (x === y) je false zato sto x i y nisu isti tip podatka
// (z == y) je false zato sto dva objekta ne mogu da se upoređuju sa ==
```

Не треба креирати логичке променљиве као објекте и поред тога што су вредности исте за оператер `==`, зато што променљиве као објекти успоравају извршавање кода и зато што имају различите врсте података за `x === y` (`boolean` и `object`), па упоређивање типа података враћа `false`. Упоређивање два идентично дефинисана објекта ће такође генерисати `false` зато што два објекта не могу да се упоређују. Када је једна променљива логичког типа, а друга објекат, тада се објекат аутоматски конвертује у логички тип, па је и упоређивање `true`.

3.4.1 *JavaScript* компарације

Компарације (*comparison operators*, упоређивања) и логички оператори се користе у *JavaScript*-у да би се добила вредност потребна за тестирање да ли је вредност тачна или није (`true` или `false`). У следећој табели су дате логичке наредбе да би се одредило да ли су промењљиве или вредности исте или различита.

```
var x = 5; // dodeljena vrednost pre uporedjivanja
```

Оператор	Опис	Упоређивање	Враћа
==	Једнако	x == 9	false
		x == 5	true
		x == "5"	true
===	Иста вредност и тип	x === 5	true
		x === "5"	false
!=	Различито	x != 9	true
!==	Није иста вредност и тип	x !== 5	false
		x !== "5"	true
		x !== 9	true
>	Веће	x > 9	false
<	Мање	x < 9	true
>=	Веће или једнако	x >= 9	false
<=	Мање или једнако	x <= 9	true

Компарације може да се користи у условним наредбама за упоређивање две вредности и предузимање акција у складу са логичком вредношћу резултата, на пример:

```
var x = 5;      // dodeljena vrednost pre uporedjivanja
var text = ""; // text je prazan string
if (x < 21) text = "Vrednost je manja"; // text postaje Vrednost je manja
```

У неким случајевима је потребно да буде испуњено више услова, истовремено или бар један од њих, и тада се користе логички оператори као у наредној табели.

```
var x = 6; // dodeljena vrednost pre uporedjivanja
var y = 3; // dodeljena vrednost pre uporedjivanja
```

Оператор	Опис	Пример	Разлог
&&	Логичко и, and	(x < 10 && y > 1) //daje true	true && true
||	Логичко или, or	(x == 5 || y == 5) //daje false	false || false
!	Логичка негација, not	!(x == y) //daje true	!(false)

Условни тројни (тернарни) оператор омогућава да се предузму две различите активности у зависности од логичке вредности услова, на пример:

```
var x = 5;              // dodeljena vrednost pre uporedjivanja
var tekst = (x < 18) ? "manje od 18":"vece od 18";
// tekst postaje manje od 18; da je x = 25 tekst postaje vece od 18
```

Истом наредбом се декларише промењљива и врши додела стринга ако је испуњен услов, `"manje od 18"`, а ако није испуњен услов тада добија другу вредност, `"vece od 18"`.

Упоређивање података који нису истог типа може да проузрокује другачије резултате од оних које очекује програмер, као што је показано у следећој табели.

Пример	Резултат	Разлог
`2 < 12`	`true`	Упоређивање бројева
`2 < "12"`	`true`	Конверзија стринга "12" у број 12
`2 < "John"`	`false`	"John" се конвертује у `NaN`
`2 > "John"`	`false`	"John" се конвертује у `NaN`
`2 == "John"`	`false`	"John" се конвертује у `NaN`
`0 == ""`	`true`	Празан стринг се конвертује у 0
`"2" < "12"`	`false`	Упоређује прве цифре, `"2" < "1"`
`"2" > "12"`	`true`	Упоређује прве цифре, `"2" > "1"`
`"2" == "12"`	`false`	Упоређује прве цифре, `"2" == "1"`

Празан стринг се конвертује у 0. Ако стринг није број, конвертује се у `NaN` а свака операција са `NaN` је `NaN`, односно резултат упоређења `NaN` са бројем је `false`. Упоређивање `2 == "John"` односно резултат упоређења `NaN` са бројем је `false`.

Упоређивање `NaN == NaN` враћа `false`, зато што `NaN` значи да је то број чија вредност није позната, па је зато и резултат `false`.

Пре компарације је пожељно да се провери да ли је променљива одговарајућег типа, па ако није да се у прегледачу испише упозорење, на пример, посматрајмо део кода:

```
x = Number(x);
if (isNaN(x)) {
  tekst = "ulazni podatak nije broj";
} else {
 tekst = (x < 18) ? "manje od 18" : "vece od 18";
}
```

Прво се тестира да ли је `x` број, па ако није приказује се текст `tekst` у прегледачу `"ulazni podatak nije broj"`, а ако јесте број, тестира се да ли је испуњен услов и у зависности од тога да ли је истинит или није, добија се приказ текста `tekst` у прегледачу `"manje od 18"` или `"vece od 18"`.

3.4.2 *JavaScript* условне наредбе

Условне наредбе (*conditional statements*) се користе да се изврше различите програмске активности у зависности од различитих услова. Условне наредбе могу да се се користе на следећи начин:

- Ако је испуњен задати услов на (`if`), извршити блок наредби.
- Ако није испуњен задати услов (`else`), извршити неки други блок наредби.
- Ако није испуњен задати услов, проверит да ли је испуњен неки други услов и ако јесте (`else if`), извршити нови блок наредби.
- Ако постоји већи број различитих блокова наредби, може се користити наредба `switch`.

Већ смо приказали пример коришћења наредбе са `if`; најпре су биле декларисане две променљиве (`x` и `text`) којима су додељене вредности, а затим је тестирана истинитост са наредбом `if`; ако услов даје резултат `true`, извршава се нова додела у наставку наредбе (`text = "Vrednost je manja"`):

```
var x = 5;      // dodeljena vrednost pre uporedjivanja
var text = ""; // text je prazan string
if (x < 21) text = "Vrednost je manja"; // text postaje Vrednost je manja
```

Да не би дошло до забуне у интерпретацији, услов је у загради (`x < 21`) а резултат услова је `true` или `false`. Без употребе сепаратора као што је зарез, довољан је бланко карактер да се раздвоји од наредбе која додељује вредност променљивој `text`. У случају да је резултат услова `false`, тада се ништа не извршава, односно променљива задржава раније додељену вредност `text = ""`.

У случају да услов има вредност `true`, често је потребно да се изврши већи број наредби, као блок наредби, па се наредбе групишу у витичасте заграде. Ако је резултат `false`, тада треба прескочити цео блок наредби као да не постоје. Због тога, најбоље је да се користи следећи запис, на пример:

```
if (x < 21) {
 text = "Vrednost je manja"; // text postaje Vrednost je manja
 x = 6; // sledeca naredba u bloku naredbi
}
```

У оваквом запису је у једном реду услов, који може да буе `true` или `false`, у следећем реду је наредба која треба да се изврши ако је услов `true`, а затим у наредним редовима може бити већи број наредби које се узвршава. Витичасте заграде групишу цео блок наредби.

Уколико условна команда `if` није написана малим словима, већ има и неко велико слово, на пример `If` или `IF`, *JavaScript* ће јавити грешку.

У претходном примеру, ако је услов `false`, команде у блоку груписаном између витичастих заграда неће бити извршене, и вредности за променљиве ће остати непромењене, `text = ""` и `x = 6`.

Уколико се жели другачија додела за услов којије `false`, потребно је додати и `else`:

```
if (x < 21) {
  text = "manje od 21"; // text postaje manje od 21
  x = 6; // sledeca naredba u bloku naredbi
} else {
 text = "vece od 21"; // text postaje vece od 21
}
```

Из примера се види да број наредби не мора да буде исти у оба блока наредби.

Уколико условна наредба даје резултат `false`, може се убацити испред `else` нови услов, коришћењем `else if`:

```
if (x < 21) {
  text = "manje od 21"; // text postaje manje od 21
  x = 6; // sledeca naredba u bloku naredbi
else if (x < 25) {
```

```
    text = "21 <= x < 25"; // text postaje 21 <= x < 25
} else {
    text = "x >= 25"; // text postaje x >= 25
}
```

Уколико је и други услов `false`, а не постоји део кода са `else`, неће бити извршена ни једна наредба, па ће у овом примеру све променљиве остати непромењене у односу на вредности пре условне команде `if`.

3.4.3 *JavaScript* `Switch` наредба

Наредба са `switch` (*switch statement*) се користи да би се извршиле различите програмске активности у зависности од више различитих услова. После сваког услова је блок наредби у витичастим заградама који се извршава ако је испуњен тражени услов (ако условни оператор враћа резултат `true`).

Наредба са `switch` се извршава само једном. За сваки наведени услов се тестира да ли је `true` и извршава се блок наредби који следи иза услове. Ако ни један услов није `true`, извршава се код који следи после `default`. Следи пример у коме је најпре декларисана променљива `dan`, променљивој је додељена вредност датума и времена `new Date(2020, 7, 15, 8, 56, 35)` или без података ако се уноси тренутно време са `new Date()`, из података се узима само број који одговара дану у недељи који може да буде број од 0 до 6. Променљива мора да се декларише пре наредбе са `var dan`, јер ће се касније у `switch`-у додељивати друге вредности променљивој. Затим се у `switch`-у тестира за све вредности који је број дана у недељи и променљивој додељује стринг са називом дана у недељи.

Да не би променљива била декларисана као број, а затим у `switch`-у се додељује стринг, боље је да после декларисања променљиве `var dan`, променљивој се не додељује вредност, с тим да се тестира у `switch`-у цео израз са

```
switch (new Date(2020, 7, 15, 8, 56, 35).getDay())
```

То збачи да и није потребно да се декларише као променљива у загради иза `switch (...)`, већ то може бити вредност или израз. Следи пример:

```
var dan = new Date(2020, 7, 15, 8, 56, 35).getDay();
switch (dan) {
  case 0:
    dan = "nedelja";
    break;
  case 1:
    dan = "ponedeljak";
    break;
  case 2:
    dan = "utorak";
    break;
  case 3:
    dan = "sreda";
    break;
  case 4:
    dan = "cetvrtak";
    break;
```

```
    case 5:
      dan = "petak";
      break;
    case 6:
      dan = "subota";
}
```

Резултат овог кода је `subota`.

3.4.3.1 `Switch` наредба и кључна реч `break`

JavaScript `switch` наредба би извршавала провере свих `case` тестова и проверавала да ли су `true`, без обзира да ли је пронађен један случај који је истинит. Да би се код ефикасније извршавао, после провере једног случаја који враћа резултат `true`, и извршења блока наредби, може да се напише кључна реч `break`, тако да се прескачу сва даља тестирања и излази из `switch` наредбе. Није потребно да се пише бр после последњег тестирања, зато што се ионако излази из `switch` наредбе па нема шта да се прескочи у извршавању. Следи пример:

```
var tekst;
switch (new Date(2020, 7, 15, 8, 56, 35).getDay()) {
  case 6:
    tekst = "Danas je subota";
    break;
  case 0:
    tekst = "Danas je nedelja";
    break;
  default:
    tekst = "Danas je radni dan";
}
```

Резултат овог кода за променљиву `tekst` је стринг `Danas je subota`.

3.4.3.2 `Switch` наредба и кључна реч `default`

JavaScript `switch` наредба не мора да има све бројеве у низу за `case` тестирање. На пример, може се тестирати услов да ли је виденд или радни дан, и који дан викенда је по датуму. За разлику од претходног примера, сада је повећана вредност датума за дан са `15` на `17`.

```
var tekst;
switch (new Date(2020, 7, 17, 8, 56, 35).getDay()) {
  case 6:
    tekst = "Danas je subota";
    break;
  case 0:
    tekst = "Danas je nedelja";
    break;
  default:
    tekst = "Danas je radni dan";
}
```

Резултат овог кода за променљиву `tekst` је стринг `Danas je radni dan`.

Исти резултат се добија ако се линије кода за случај `default` преместе на друго место, на пример на почетак:

```
var tekst;
switch (new Date(2020, 7, 17, 8, 56, 35).getDay()) {
  default:
    tekst = "Danas je radni dan";
    break;
  case 6:
    tekst = "Danas je subota";
    break;
  case 0:
    tekst = "Danas je nedelja";
}
```

Сада последња линија `switch` наредбе за случај `case 0:` нема кључну реч `break`, али случај са `default:` има кључну реч `break` у блоку који се извршава за `default`.

3.4.3.3 `Switch` наредба и заједнички кодни блокови

JavaScript `switch` наредба може да има више заједничких случајева, на пример:

```
var tekst;
switch (new Date(2020, 7, 17, 8, 56, 35).getDay()) { case 5:
    tekst = "predstoji vikend";
    break;
 default:
    tekst = "radna nedelja";
    break;
  case 0:
  case 6:
    tekst = "sada je vikend";
}
```

У случајевима `case 0:` и `case 6:`, вредност стринга постаје `sada je vikend`. Петак је посебно издвојен као дан који претходи викенду, а остали случајеви су за радне дане.

3.4.3.4 `Switch` наредба и стриктна компарација

JavaScript `switch` користи стриктну компарацију (*strict comparison*), (`===`), што значи да тестира и вредност и тип података. На пример:

```
var x = "0";
switch (x) {
  case 0:
    tekst = "iskljuceno";
    break;
  case 1:
    tekst = "ukljuceno";
    break;
  default:
   tekst = "nije pronadjena vrednost";
}
```

Иако је променљива x типа стринг која садржи број 0, и компарација стринга "0" коришћењем оператора (==) са бројем 0 даје true, па би стринг tekst добио вредност "iskljuceno", у switch наредби се користи (===), компарација стринга "0" коришћењем оператора (===) са бројем 0 даје false, па ће резултат бити под default, односно "nije pronadjena vrednost".

3.4.4 Питања и задаци за вежбу

1. Које су могуће логичке вредност у *JavaScript*-у?
2. Како се програмира компарација са логичким операторима?
3. Шта су условне наредбе и како се се користе у *JavaScript*-у?
4. Чему служи и како се користи наредба switch?
5. Објаснити на примеру како се користи кључна реч break.
6. Где у блоку switch наредби може да буде кључна реч default?
7. Шта се користи за стриктну компарацију?

3.5 *JavaScript* програмске петље

Резиме

Циљ овог поглавља је да студент савлада коришћење *JavaScript* програмских петљи за програмирање читљивог кода и ефикасно извршавање операција са великим бројем података.

Увод у *JavaScript* програмске петље

JavaScript програмске петље (*loops*, петље) омогућавају да се више пута изврши блок са програмским наредбама. Петље су погодне за случајеве када је потребно да се исти блок наредби изврши више пута, за различите вредности, при чему се најчешће не зна унапред колико пута треба извршити блок наредби из петље. На овај начин код постаје разумљивији и измене у коду су једноставније уместо да се на више места раде измене. Петље се најчешће користе када се ради са низовима. На пример, посматрајмо низ:

```
var osobe = ["Miro", "Branka", "Ana", "Ema"];
// originalni niz ["Miro", "Branka", "Ana", "Ema"]
var tekst = "dosli su ";
tekst += osobe [0] + " "; // dosli su Miro
tekst += osobe [1] + " "; // dosli su Miro Branka
tekst += osobe [2] + " "; // dosli su Miro Branka Ana
tekst += osobe [3] + " "; // u pregledacu: dosli su Miro Branka Ana Ema
```

Уместо појединачне доделе сваког елемента низа osobe у стринг tekst, боље је да се користи петља:

```
var osobe = ["Miro", "Branka", "Ana", "Ema"];
var tekst = "dosli su ";
var i;
for (i = 0; i < osobe.length; i++) {
  tekst += osobe[i] + " ";
}                          // u pregledacu: dosli su Miro Branka Ana Ema
```

Када се појединачно додељују елементи низа, програмер мора да зна дужину низа. Низу могу да се додељују нови елементи, или избацују елементи из низа, па код не може да се извршава аутоматизовано без грешака.

Када се користи петља, декларишу се све променљиве пре петље (`osobe`, `tekst`, `i`), у петљи се извршавају наредбе додавања у стринг елемената низа (`tekst +=osobe[i]+" "`) од првог до последњег елемента низа (`i=0`, `i++`), и петља се извршава онолико пута колико је дужина низа `osobe.length`. На овај начин ће код увек радити исправно без обзира да ли је дошло до промене броја елемената низа пре извршавања петље.

Постоји више врста петљи:

- `for` петља која пролази кроз блок наредби већи број пута
- `for/in` петља кроз особине објекта
- `for/of` петља кроз вредности објекта који може да буде са итерацијама
- `while` петља кроз блок наредби све дотле док специкициран услов има вредност `true`
- `do/while` - петља кроз блок наредби све дотле док је вредност специфицираног `true`

3.5.1 *JavaScript for* петља

Петља `for` има следећу синтаксу:

```
for (iskaz1; iskaz2; iskaz3) {
  naredbeKojeSeIzvrsavaju;
}
```

На пример,

```
for (i = 0; i < 5; i++) {
 tekst += "vrednost broja je " + i + "<br>";
}
```

`iskaz1` (исказ, *statement*) се извршава само једном пре блока наредби из петље. `i = 0`.

`iskaz2` дефинише услов за извршавање блока наредби, `i < 5`.

`iskaz3` се извршава сваки пут након што се блок наредби изврши.

У примеру, прво се постави иницијална вреднос промењиве на 0. Проверава се услов да ли је `true`, и у примеру све док је `i` мање од 5 извршава се блок наредби који је између витичастих заграда. Након завршетка блока наредби, извршава се трећи исказ а то је да се `i` инкрементира за 1. Поново се проверава услов који је `true` све док је `i` добија вредности 0, 1, 2, 3, 4. Ако после инкрементирања `i` добије вредност 5, услов постаје `false` и програм излази из петље.

У следећем примеру се пре петље декларишу све промењиве без доделе вредности за промењиве које се користе у петљи. Све иницијализације се смештају у `iskaz1`.
`(i = 0, len = osobe.length, tekst = "dosli su ")`

Други исказ, `iskaz2`, (`i < len`), садржи услов који користи промењиве са додељеним вредностима у `iskaz1`.

Трећи исказ (`iskaz3`, `i++`) садржи итератор у петљи који се извршава после блока наредби. На пример:

```
var osobe = ["Miro", "Branka", "Ana", "Ema"];
var tekst , i, len;
for (i = 0, len = osobe.length, tekst = "dosli su "; i < len; i++) {
  tekst += osobe[i] + " ";
}                     // u pregledacu: dosli su Miro Branka Ana Ema
```

Први исказ може да се изостави ако је иницијализација извршена пре петље. На пример:

```
var osobe = ["Miro", "Branka", "Ana", "Ema"];
var tekst = "dosli su ";
var len = osobe.length;
var i = 0;
for (; i < len; i++) {
  tekst += osobe[i] + " ";
} // u pregledacu: dosli su Miro Branka Ana Ema
```

У првој линији петље остаје тачка-зарез `;`, али испред нема исказа. Услов је прегледнији јер је променљива декларисана са додељеном вредношћу пре петље.

У петљи није неопходан ни други услов, `iskaz2`, ако се обезбеди да се у блоку наредби петље обезбеди излаз из петље, на пример коришћењем кључне речи `break`.

Ако се не обезбеди излазак из петље, преко услова или са `break`, петља ће се бесконачно извршавати.

Ни трећи исказ није обавезан ако се промена вредности итератора обезбеди у петљи, али је увек обавезан карактер тачка-зарез `;`. На пример:

```
var osobe = ["Miro", "Branka", "Ana", "Ema"];
var tekst = "dosli su ";
var len = osobe.length;
var i = 0;
for (; i < len;) {
  tekst += osobe[i] + " ";
  i++;
} // u pregledacu: dosli su Miro Branka Ana Ema
```

У петљи није неопходан ни један од исказа, `iskaz1`, `iskaz2` или `iskaz3`, ако је `iskaz1` дефинисан пре петље, ако је `iskaz2` у блоку команди за излазак из петље, ако је `iskaz3` у петљи тако да се мења вредност која се користи за тестирање за испуњење услова изласка из петље.

3.5.2 *JavaScript for/in* петља

JavaScript петља `for/in` пролази кроз особине објекта, на пример:

```
var osobe = {ime:"Miro", prezime:"Lu", godine:63};

var tekst = "nastavnik je ";
var x;
for (x in osobe) {
  tekst += osobe[x] + " ";
} // u pregledacu: nastavnik je Miro Lu 63
```

Елементи низа се добијају по индексу, а када је објекат, тада се вредностима објекта приступа по асоцијативним вредностима.

3.5.3 *JavaScript for/of* петља

Петља `for/of` пролази кроз итеративне вредности неког објекта. Ова петља омогућава да се у петљи користе подаци из структура које су итерабилне, као што су низови, стрингови, *Maps*, *NodeLists*. Синтакса `for/of` петље је следећа:

```
for (promenljiva of iterabilniPodatak) {
 // blok naredbi koji se izvrsava
}
```

Промењљива `promenljiva` добија вредност за сваку наредну итерацију следеће особине. Промењљива може да се декларише са `const`, `let` или `var`. Податак `iterabilniPodatak`, је објекат који има итерабилну особину. На пример:

```
var osobe = ['Miro', 'Branka', 'Ana', 'Ema'];
var x;
for (x of osobe) {
  document.write(x + "<br >");
}
```

Као резултат се добија у прегледачу:

```
Miro
Branka
Ana
Ema
```

За стринг може да се тестира следећи пример:

```
var txt = 'JavaScript za IP';
var x;

for (x of txt) {
  document.write(x + "<br>");
}
```

Као резултат се добија у прегледачу:

```
J
a
v
a
S
c
r
i
p
t

z
a
```

```
I
P
```

3.5.4 *JavaScript while* петља

Петља `while` може да изврши блок наредби све док постављени услов има вредност `true`.

Синтакса `while` петље је следећа:

```
while (uslovniIskaz) {
 // blok naredbi koji se izvrsava
}
```

Условна наредба (коју ћемо назвати условни исказ, `uslovniIskaz`) израчунава тачност исказа који има као резултат једну од две вредности `true` или `false`. У случају да је вредност `true`, извршава се блок наредби између витичастих заграда. На пример:

```
var i = 0;
var tekst = "";
while (i < 5) {
 tekst += " <br> broj je " + i;
  i++;
}
```

Пре извршавања петље потребно је да се декларишу променљиве са `var` и доделе вредности које се користе у петљи, `i = 0`, `tekst = ""`. У петљи је постављен услов `(i < 5)` да се извршава ако је вредност услова `true`. У самој петљи се генерише стринг `tekst` који се приказује у параграфу. Последња наредба је инкремент променљиве, `i++`, која се користи за тестирање тачности услова.

У прегледачу се приказује:

```
broj je 0
broj je 1
broj je 2
broj je 3
broj je 4
```

У случају да се у петљи не мења вредност променљиве која се користи за условни исказ, петља неће имати крај што може довести до прекида рада прегледача.

Петља `do/while` може да изврши блок наредби све док постављени услов има вредност `true`. За разлику од `while` петље, петља `do/while` извршава блок наредби бар једном. На крају петље се тестира услов и тек се тада проверава да ли условна наредба (`uslovniIskaz`), има вредност `true`. Ако је резултат `true`, извршава се блок наредби још једном, а ако је резултат `false`, извршавају се наредбе које су после петље `do/while`.

Синтакса `do/while` петље је следећа:

```
do {
 // blok naredbi koji se izvrsava
}
while (uslovniIskaz)
```

Условна наредба (`uslovniIskaz`) тестира се на крају блока наредби. На пример:

```
var i = 0;
var tekst = "";
do {
 tekst += " <br> broj je " + i;
  i++;
}
while (i < 5)
```

У прегледачу се приказује:

```
broj je 0
broj je 1
broj je 2
broj je 3
broj je 4
```

У случају да се у петљи не мења вредност промењиве која се користи за условни исказ, петља неће имати крај што може довести до прекида рада прегледача.

У примерима за `while` и `do/while`, број је додаван у стринг који треба да се прикаже у прегледачу. Међутиом, промењива `i` може да се користи као индекс за `osobe[i]`, и тада се у стринг додају елементи низа. То значи да код са `for` и `while` може да генерише исти резултат, осим што за `for` нису потребни искази `iskaz1` и `iskaz3`. На пример:

```
var osobe = ["Miro", "Branka", "Ana", "Ema"];
var i = 0;
var tekst = "";

for (;osobe[i];) {
  tekst += osobe [i] + "<br>";
  i++;
}
```

Петља се извршава све дотле док постоје елементи низа, а услов за извршавање постаје `false` кад индекс постане једнак дужини низа. У петљи услов за излазак из петље је исти, а то је када не постоји елемент у низи за индекс `i` једнак дужини низа `i=osobe.length`:

```
while (osobe[i])
```

Последњи примери показују концизност и разумљивост кода.

3.5.5 *JavaScript break* и *continue* у петљама

Наредба `break` прекида извршавање кода и излази из петље (`for`, `for/in`, `for/of`, `while`, `do/while`, или `switch` наредбе. Наредба `break` прекида петљу и наставља код који је после петље, ако постоји. На пример:

```
var tekst = "";
for (i = 0; i < 5; i++) {
  if (i === 3) { break; }
 tekst += " <br> broj je " + i;
}
```

Резултат у прегледачу који приказује стринг `tekst` је:

```
broj je 0
broj je 1
broj je 2
```

Након што је променљива добила вредност 3, прекида се извршавање кода и у прегледачу се приказују бројеви од 0 до 2. Без линије кода у којој је `break` били би приказани бројеви од 0 до 4.

Наредба `continue` прекида извршавање једне итерације петље, ако се деси спецификовани случај, и прелази на следећу итерацију петље. На приме:

```
var tekst = "";
for (i = 0; i < 5; i++) {
  if (i === 3) { continue; }
 tekst += " <br> broj je " + i;
}
```

У прегледачу се исписују сви бројеви осим броја 3.

```
broj je 0
broj je 1
broj je 2
broj je 4
```

3.5.6 *JavaScript* наредба *label*

Наредба `label` се користи да означи име линије кода у којој се налазе наредбе којима се наставља извршавање програма, на пример:

```
label:
statements
```

Наредбе `break` и `continue` statements су једине наредбе преко којих се прескочити део кода да би се дошло до блока команди које треба извршити, на пример:

```
break labelname;

continue labelname;
```

Наредба `continue` (са или без ознаке лабеле) може да се користи да се прескочи једна итерација у петљи.

Наредба `break`, без означавања лабеле може да се користи за излазак из петље или команде `switch`.

Са ознаком лабеле, наредна `break` може да се користи на било који блок наредби. На пример:

```
var osobe = ["Miro", "Branka", "Ana", "Ema"];
var tekst = "prijatelji ";
oznaka: {
  tekst += osobe[0] + "<br>";
  tekst += osobe[1] + "<br>";
  break oznaka;
  tekst += osobe[2] + "<br>";
```

```
        tekst += osobe[3] + "<br>";
    }
    tekst += "kraj <br>";
```

Резултате приказа стринга у прегледачу је `tekst`:

```
prijatelji Miro
Branka
kraj
```

Прво је урађено декларисање промењљивих и додела почетних вредности. Лабела је место које означава блок команди. Након извршавања команди између витичастих заграда, извршавају се две наредбе, тако што се стрингу `tekst` додељују два елемента низа. Са `break` се излази из блока наредби, иде на лабелу и извршава наредба после блока у витичастим заградама. У прегледачу се исписује стринг генерисан овим кодом.

3.5.7 Питања и задаци за вежбу

1. Зашто се користе *JavaScript* програмске петље?
2. Написати синтаксу *JavaScript* `for` петље.
3. Написати синтаксу *JavaScript* `for/in` петље.
4. Написати синтаксу *JavaScript* `for/of` петље.
5. Написати синтаксу *JavaScript* `while` петље.
6. Објаснити како се користе *JavaScript* `break` и `continue` у *JavaScript* петљама.
7. Како се и зашто користи наредба label у *JavaScript* програмским петљама?

3.6 *JavaScript* типови конверзија података

Резиме: Циљ овог поглавља је да студент разуме аутоматске конверзије података како би избегао могуће погрешне интерпретације.

У *JavaScript*-у се користи `Number()` за конверзију податка у број, `String()` за конверзију податка у стринг, `Boolean()` за конверзију податка у логичку вредност.

У *JavaScript*-у постоји 5 различитих типова података који садрже вредности:

- стринг
- број
- логичка вредност
- објекат
- функција

Постоји 6 типова објеката:

- објекат као `Object`,
- објекат за датум и време као `Date`,
- објекат за низ као `Array`,
- објекат за стринг као `String`,
- објекат за број као `Number`,
- објекат за логичку вредност као `Boolean`.

Постоје 2 типа података који не садрже вредности:

- `null`,
- `undefined`.

За одређивање типа податка може да се користи `typeof`:

```
typeof "Miro"                 // vraca "string"
typeof 3.14                   // vraca "number"
typeof NaN                    // vraca "number"
typeof false                  // vraca "boolean"
typeof [1,2,3,4,5]            // vraca "object"
typeof {ime:'Miro', god:64}   // vraca "object"
typeof new Date()             // vraca "object"
typeof function () {}         // vraca "function"
typeof mojPrijatelj           // vraca "undefined" *
typeof null                   // vraca "object"
```

Важно је уочити следеће чињенице:

- Тип података `NaN` је број, то је број којем се не зна вредност,
- Тип података за низ је објекат,
- Тип података за датум и време је објекат,
- Тип података `null` је објекат,
- Тип података промењиве која је undefined је `undefined`
- Тип података промењиве која је декларисана, али јој није додељена вредност, је `undefined`.

Не може се користити `typeof` да би се сазнало да је *JavaScript* објекат у ствари низ или податак за датум и време.

Оператори као што су `+`, `-`, `*`, `/` немају тип података. Оператор је и `typeof`, и није промењљива па зато не може ни да буде тип података. Међутим, то увек враћа као резултат стринг који садржи тип операнда на који се примењује оператор.

Особина конструктора враћа функцију конструктора (`constructor`) за све *JavaScript* промењиве. На пример:

```
"Miro".constructor                // vraca function String()  {[native code]}
(3.14).constructor                // vraca function Number()  {[native code]}
false.constructor                 // vraca function Boolean() {[native code]}
[1,2,3,4,5].constructor           // vraca function Array()   {[native code]}
{ime:'Miro', god:64}.constructor  // vraca function Object()  {[native code]}
new Date().constructor            // vraca function Date()    {[native code]}
function () {}.constructor        // vraca function Function(){[native code]}
```

Провером особине конструктора може се наћи да ли је неки објекат низ (`Array`) тако што садржи реч `"Array"`:

```
var prijatelji = ["Miro", "Branka", "Ana", "Ema"];
var x = isArray(prijatelji);    // vraca true

function isArray(myArray) {
  return myArray.constructor.toString().indexOf("Array") > -1;
}
```

У прегледачу се добија логичка вредност `true` ако је променљива `prijatelji` низ.

Једноставније је да се провери да ли је објекат `Array function`:

```
var prijatelji = ["Miro", "Branka", "Ana", "Ema"];
var x = isArray(prijatelji);    // vraca true
var stringP = '["Miro", "Branka", "Ana", "Ema"]';
var x = isArray(stringP);       // vraca false

function isArray(myArray) {
  return myArray.constructor === Array;
}
```

У овом примеру постоје две променљиве, једна је типа низ, `prijatelji`, друга је типа стринг, `stringP`; једноструки наводи додељују вредност стринг који личи на низ `prijatelji`.

Провером особине конструктора може се наћи да ли је неки објекат за датум и време (`Date`, садржи реч `"Date"`):

```
var myDate = new Date();
var x = isDate(myDate);       // vraca true

function isDate(myDate) {
  return myDate.constructor.toString().indexOf("Date") > -1;
}
```

Једноставније је да се провери да ли је објекат `Date function`:

```
var myDate = new Date();
var x = isDate(myDate);       // vraca true

function isDate(myDate) {
  return myDate.constructor === Date;
}
```

JavaScript променљиве могу да се конвертују у нову променљиву или други тип података на два начина:

- Коришћењем *JavaScript* функције.
- Аутоматски по правилима *JavaScript*-а.

За конверзију броја у стринг користи се глобална метода `String()` за конверзију бројева у стринг. Може да се користи за било који тип бројева, константи, променљивих или израза, на пример:

```
var x = 123;             // promenljiva x tipa broj
String(x)                // vraca string 123 iz promenljive x
String(123)              // vraca string 123 iz konstante 123
String(100 + 23)         // vraca string 123 iz izraza (100 + 23)
```

Исто може да се уради коришћењем методе `toString()` која враћа број као стринг:

```
var x = 123;             // promenljiva x tipa broj
x.toString();            // vraca string 123 iz promenljive x
(123).toString();        // vraca string 123 iz konstante 123
(100 + 23).toString();   // vraca string 123 iz izraza (100 + 23)
```

Методе које могу да се користе за конверзију бројева у стрингове дате су у следећој табели:

Метода (Method)	Опис
toExponential()	Враћа стринг са бројем заокруженим и приказаним у експоненцијалној нотацији.
toFixed()	Враћа стринг са бројем заокруженим и приказаним са специфікованим бројем децимала.
toPrecision()	Враћа стринг са бројем приказаним на одређену дужину.

Глобална метода String() може да конвертује и логичке вредности у стринг:

```
String(false)      // vraca "false"
String(true)       // vraca "true"
```

Исто може да се уради коришћењем методе toString() која враћа логичку вредност као стринг:

```
false.toString()   // vraca "false"
true.toString()    // vraca "true"
```

Глобална метода String() може да конвертује објекте типа Date() у стринг:

```
String(Date())  // vraca W. Europe Daylight Time
```

Исто може да се уради коришћењем методе toString():

```
Date().toString()  // vraca W. Europe Daylight Time
```

Више о методама за време и датуме се може наћи у *Date Methods*.

Глобална метода Number() може да конвертује стрингове у бројеве. Стрингови који се састоје од цифара могу да се конвертују у бројеве. Празан стринг се конвертује у број 0. Све друго се конвертује у NaN (*Not a Number*).

```
Number("10.33");        // vraca broj 10.33
Number(" 8.656e-1 ");   // vraca broj 0.8656
Number(" ");            // vraca broj 0
Number("");             // vraca broj 0
Number("10 33");        // vraca NaN tipa broj
Number("Miro");         // vraca NaN tipa broj
```

Методе које могу да се користе за конверзију стрингова у бројева дате су у следећој табели:

Метода (Method)	Опис
parseFloat()	Парсује стринг и враћа број у покретном зарез (тачки)
parseInt()	Парсује стринг и враћа цео број

Унарни оператор + може да конвертује променљиву у број:

```
var y = "5";      // y je string
var x = + y;      // x je broj
```

Ако променљива не може да се конвертује у број, ипак ће постати број али ће имати вредност `NaN` (*Not a Number*, број за који се не зна која је вредност):

```
var y = "Miro";    // y je string
var x = + y;       // x je broj NaN
```

Глобална метода `Number()` може да конвертује логичку вредност у број:

```
Number(true);          // vraca broj 1
Number(false);         // vraca broj 0
```

Глобална метода `Number()` може да конвертује објекат типа `Date` у број:

```
d = new Date();
Number(d)          // vraca 1597474595000
```

Метода `.getTime()` може да се користи да се добије исти резултат:

```
d = new Date();
d.getTime()        // vraca 1597474595000
```

Аутоматски типови конверзије могу да произведу резултат који није увек како програмер очекује. *JavaScript* покушава да претпостави шта је програмер хтео у ситуацији када је наишао на неочекивани тип података и покушаће да врати исправан тип података:

```
5 + null    // vraca 5         null se konvertuje u 0
"5" + null  // vraca "5null"   null se konvertuje u "null"
"5" + 2     // vraca "52"      2 se konvertuje u "2"
"5" - 2     // vraca 3         "5" se konvertuje u 5
"5" * "2"   // vraca 10        "5" i "2" se konvertuju u 5 i 2
```

JavaScript аутоматски позива функцију `toString()` када треба да се добије приказ у прегледачу за објекат или променљиву:

```
document.getElementById("demo").innerHTML = myVar;

// if myVar = {ime:"Fmiro"} // toString konvertuje u "[object Object]"
// if myVar = [1,2,3,4,5]   // toString konvertuje u "1,2,3,4,5"
// if myVar = new Date()  // toString konvertuje u " Sat Aug 15 2020 ..."
```

Бројеви и логичке вредности се такође аутоматски конвертују при приказу у прегледачу, само што се то не може јасно уочити зато што се не приказују знаци навода:

```
// if myVar = 123        // toString konvertuje u "123"
// if myVar = true       // toString konvertuje u "true"
// if myVar = false      // toString konvertuje u "false"
```

У следећој табели су дате аутоматске конверзије вредности, у бројеве, стрингове и логичке вредности:

Оригинална вредност	Конверзија у број	Конверзија у стринг	Конверзија у логичку вредност
false	0	"false"	false
true	1	"true"	true
0	0	"0"	false

1	1	"1"	true
"0"	0	"0"	**true**
"000"	0	"000"	**true**
"1"	1	"1"	true
NaN	NaN	"NaN"	false
Infinity	Infinity	"Infinity"	true
-Infinity	-Infinity	"-Infinity"	true
""	**0**	""	**false**
"20"	20	"20"	true
"twenty"	NaN	"twenty"	true
[ ]	**0**	""	true
[20]	**20**	"20"	true
[10,20]	NaN	"10,20"	true
["twenty"]	NaN	"twenty"	true
["ten","twenty"]	NaN	"ten,twenty"	true
function(){}	NaN	"function(){}"	true
{ }	NaN	"[object Object]"	true
null	**0**	"null"	false
undefined	NaN	"undefined"	false

Вредности које имају наводнике означавају стрингове. **Црвено су означене вредности које неки програмери не очекују као аутоматску конверзију а што може довести до грешке**.

3.6.1 Питања и задаци за вежбу

1. Проверити све примере конверзије из овог поглавља?
2. Објаснити зашто неки програмери погрешно интерпретирају аутоматску конверзију која се дешава у *JavaScript*-у за сваки пример означен црвом бојом из претходне табеле?

3.7 *JavaScript JSON*

Резиме: Циљ овог поглавља је да се студент оспособи за коришћење *JSON* формата.

У *JavaScript*-у *JSON* је формат за складиштење и пренос података. *JSON* се често користи за слање података са сервера до клијента за приказ у прегледачу на веб страни.

У *JavaScript*-у за *JSON* се каже да је:

- Скраћени назив за ***J**ava**S**cript **O**bject **N**otation*.
- Једноставан формат за размену података.
- Језички независан.
- Једноставан за разумевање и разумљив за опис података.

Иако је синтакса *JSON*-а изведена из *JavaScript* објектне нотације, када се каже да је језички независан то значи да се може користити и у другим програмским језицима пре свега зато што је у текстуалном формату. Код за читање и генерисање *JSON* података може да се уради у било ком програмском језику.

У следећем примеру, *JSON* синтакса дефинише објекат за запослене, низ који садржи три записа о запосленима:

```
{
"zaposleni":[
  {"ime":"Miro", "prezime":"Lu"},
  {"ime":"Ana", "prezime":"Smit"},
  {"ime":"Ema", "prezime":"Bond"}
]
}
```

Синтаксно, *JSON* формат је идентичан коду за генерисање *JavaScript* објекта. Стога се *JSON* подаци једноставно конвертују у *JavaScript* објекте.

Синтакса *JSON* записа је следећа:

- Подаци су у паровима назив:вредност.
- Подаци су раздвојени зарезима.
- Витичасте заграде се користе за груписање објаката.
- Угласте заграде се користе за груписање низова.

JSON подаци су у паровима назив:вредност као и када се генеришу објекти у *JavaScript*-у.

Пар назив:вредност се састоји од поља за назив, затим следи двотачка `:`, а после двотачке је вредност. Назив поља је увек између двоструких наводника, за разлику од *JavaScript* стринга који може да буде и између једноструких наводника, на пример:

```
"ime":"Miro"
```

JSON објекат се записује између витичастих заграда. Као и код *JavaScript*-а, објекат може да садржи више парова назив:вредност:

```
{"ime":"Miro", "prezime":"Lu"}
```

JSON низ се записује између угластих захрада. Као и у *JavaScript*-у, елемент низа може да буде објекат:

```
"zaposleni":[
  {"ime":"Miro", "prezime":"Lu"},
  {"ime":"Ana", "prezime":"Smit"},
  {"ime":"Ema", "prezime":"Bond"}
]
```

У овом примеру је пар назив:вредност такав да је назив `"zaposleni"`, а после двотачке је вредност која је низ `[...]` који садржи три елемента `[{}, {}, {}]`. Сваки елемент низа је објекат који се састоји од парова који су раздвојени зарезима, на пример `"ime":"...", "prezime":"..."`.

JSON као текстуални запис у форми стринга који се са стране сервера шаље за приказ на веб страни потребно је да се конвертује у *JavaScript* објекат. Уместо слања *JSON* записа, биће генерисан стринг као конкатенација већег броја мањих стрингова који чине *JSON* запис у *JSON* синтакси.

```
var tekst = '{ "zaposleni" : [' +
'{ "ime":"Miro" , "prezime":"Lu" },' +
```

```
'{ "ime":"Ana" , "prezime":"Smit" },' +
'{ "ime":"Ema" , "prezime":"Bond" } ]}';
```

У *JavaScript* коду декларисана је променљива `tekst`, којој је дедељен већи број стринкова. Обзиром да се у стринговима појављују двоструки наводници ", за доделу и конкатенацију су стрингови означени једноструким наводницима '. Стрингови који се генеришу имају и одговарајуће заграде, тако да су појединачни стрингови:

```
'{ "zaposleni" : ['
'{ "ime":"Miro" , "prezime":"Lu" },'
'{ "ime":"Ana" , "prezime":"Smit" },'
'{ "ime":"Ema" , "prezime":"Bond" } ]}'
```

Сада је потребно из једне променљиве `tekst` која садржи текстуални стринг у *JSON* формату, издвојити стрингове као *JavaScript* објекте. Конверзија се ради са *JavaScript* уграђеном (*built-in*) функцијом `JSON.parse()`, на пример:

```
var objekatJson = JSON.parse(tekst);
```

Део кода који издваја из текстуалног записа и *JSON* формату одговарајуће објекте је дат у примеру:

```
<p id="demo"></p>
<script>
  var tekst = '{ "zaposleni" : [' +
  '{ "ime":"Miro" , "prezime":"Lu" },' +
  '{ "ime":"Ana" , "prezime":"Smit" },' +
  '{ "ime":"Ema" , "prezime":"Bond" } ]}';
  var objekatJson = JSON.parse(tekst);
  document.getElementById("demo").innerHTML =
  objekatJson.zaposleni[1].ime + " " +
  objekatJson.zaposleni[1].prezime;
</script>
```

У прегледачи ће бити приказан пасус у коме пише:

```
Ana Smit
```

Индекс 1 означава други елемент низа који садржи објекат `zaposleni`, из кога се издвајају особине објекта `ime` и `prezime` и затим између њих убацује стринг са бланко карактером " ", да би био формиран стринг за приказ у прегледачу који исписује име, бланк знак и презиме запосленог, `Ana Smit`.

3.7.1 Питања и задаци за вежбу

1. Да ли су подаци у *JSON* формату читљиви при преносу преко интернета?
2. Објаснити сличности и разлику између дефинисања објекта у *JavaScript*-у и *JSON* формата.

4 Структура сервера и сервлета

Резиме

Циљеви овог поглавља су да се студенти оспособе за инсталирање софтвера за потребе коришћење сервлета у веб апликацијама и да студент разуме функционисање сервлета израдом једноставних примера.

4.1 Структура сервера за сервлете

Резиме

Циљ овог поглавља је да се студенти оспособе за инсталирање софтвера за потребе коришћење сервлета у веб апликацијама. Сви поступци су објашњени кроз примере по систему корак-по-корак.

Увод у

Тема у овом поглављу намењена је студентима као прва практична реализација сервлета у којој студенти треба да разумеју како изгледа структура сервера, где се налазе фајлови за имплементацију сервлета, и како да пренесу тестирану верзију са клијентског рачунара на серверски рачунар.

Пре него што се студент посвети сервлетима и другим динамичким апликацијама, пожељно је да се студент подсети неких карактеристике ових апликација, почев од инсталирања сервера до тестирања и верификације програма са сервлетима.

4.1.1 Инсталирање и деинсталирање сервера на клијентском рачунару

У претходним поглављима је показано како се инсталира сервер. Међутим, то није инсталација која се уобичајено ради на рачунарима, где се извршни фајл користи да се стартује поступак инсталације, затим неколико пута кликне дугме *Next*, и на крају дугме *Finish*, када се најчешће појављује иконица апликације на десктопу, а инсталирани

програм се налази у *Program Files*. Када се жели деинсталирање таквог програма, пронађе се поље *uninstall*, и стартује деинсталирање програма.

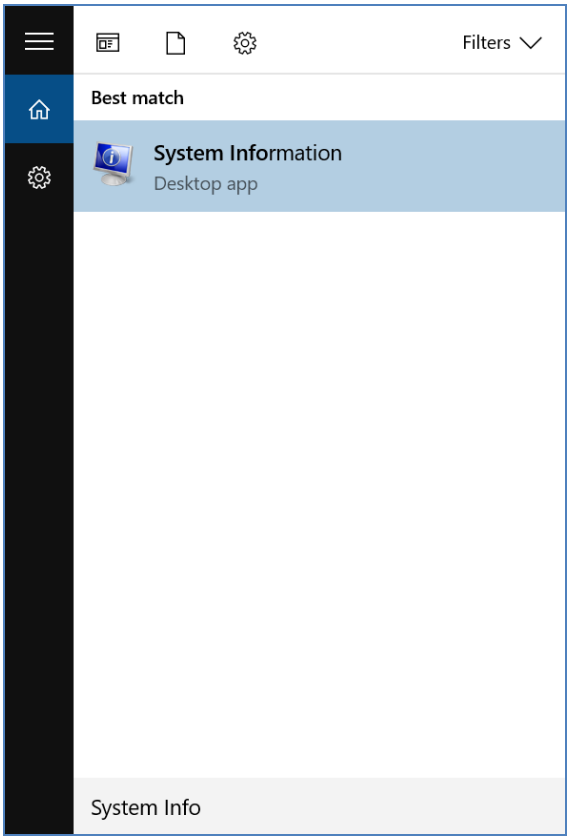
Серверски фајлови су деархивирани (распаковани из архиве која је преузета са Интернета), смештени у фолдер који сами бирамо (а то није у *Program Files*), нема кликтања по дугмићима *Next* и *Finish*, и нема потребе да се стартује *uninstall*. Довољно је обрисати фолдер у коме је био инсталиран сервер (пазити да не обришете фајлове које сте сами направили, и желите да их сачувате).

Када се рачунар ресетује, или искључи па укључи, ако нема фолдера где је био инсталиран сервер или нема серверских фајлова, неће бити могуће да се покрене рад сервера, али ништа се неће догодити што се тиче других апликација на рачунару.

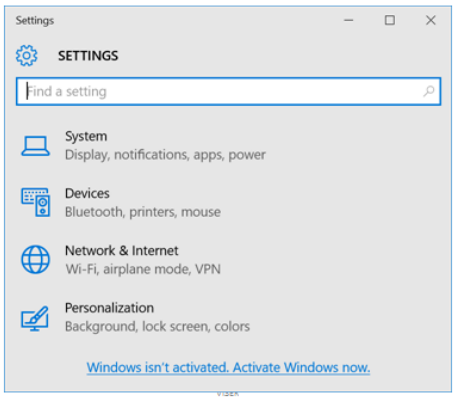
Комплетно све што је било на серверу може да се пребаци у други директоријум, на други рачунар на коме ће све функционисати ако смо задржали путање до серверских и сопствених фајлова.

Ако се комплетан садржај пребацује на други рачунар, осим путања до фајлова, треба водити рачуна и да ли су карактеристике рачунара компатибилне. Зато се предлаже следећа процедура за идентификовање компатибилности рачунара када је сервер у питању.

Важно је уочити да ли рачунар, на који се преноси фолдер са серверским фајловима, има исти оперативни систем, и ако нема, потребна је нова инсталација сервера за одговарајући оперативни систем. Други детаљ који је важан код инсталације јесте да треба проверити да ли је рачунарска магистрала са 32 или 64 бита. На слици 4.1.1 је показано како се преко *Search* (у доњем левом углу таскбара на иконицу која личи на лупу) тражи кључна реч *System Info*. Нуди се опција *System Information*, где се налазе подаци о хардверу рачунара.



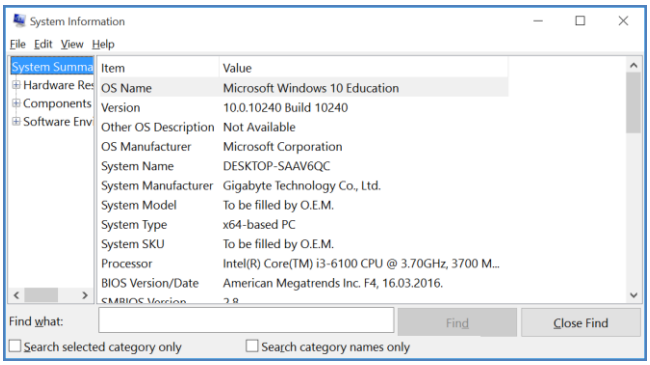
Слика 4.1.1 Преко *Search* се тражи кључна реч *System Info*



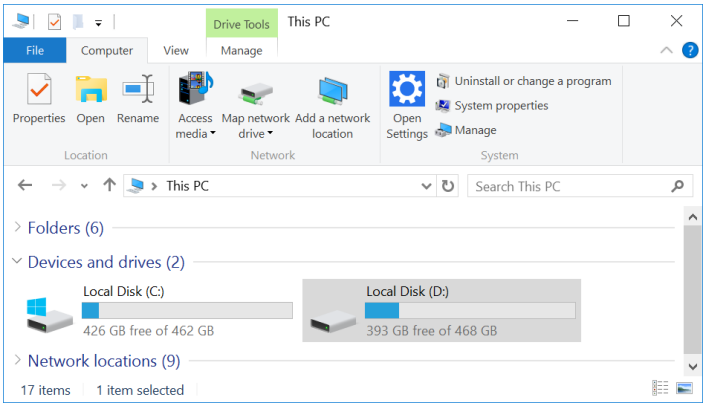
Слика 4.1.2 Са *System Info* се иде на *Settings*, па на *System*

Према слици 4.1.2, кликне се на *Settings*, па на *System*, и добија се приказ као на слици 4.1.3. У приказу података за *System Information*, препознајемо *System Type*, где пише *x64*, *OS name Windows* 10. То значи да за овај рачунар знамо који је инсталирани оперативни систем и колико бита користи процесор за рачунарску реч.

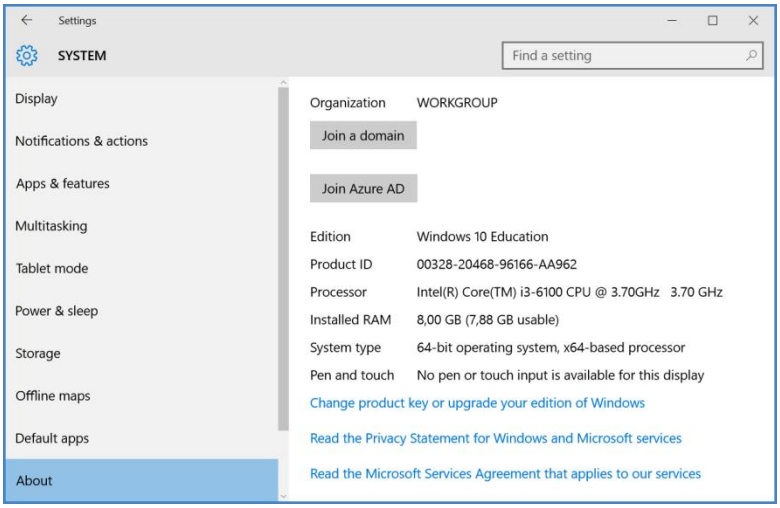
Алтернативни начин је да се преко било ког отвореног фолдера, на пример као на слици 4.1.4, кликне у левој колони на *This PC* (ако је селектован *Navigation Pane*), па кликне на *Computer* (одмах до *File*), па се кликне на *Open Settings*, па изабере језичак *About*, након чега се добија слика 4.1.5. Поново добијамо информацију о оперативном систему (*Windows* 10) и броја бита које користи процесор (64).



Слика 4.1.3 Садржај *System Information*



Слика 4.1.4 Преко *This PC*, *Computer*, *Open Settings*, *About*



Слика 4.1.5 Подаци о рачунару на језичку *About*

На неком другом рачунару или оперативном систему треба поновити ове кораке док се не дође до података који су важни за избор сервера на рачунару.

Као закључак овог дела, уочавамо да је важно да се познају основне карактеристике рачунара како би серверски фајлови бити компатибилни са радом рачунара. Серверски фајлови се не могу деинсталирати, већ се једноставно обришу када више нису потребни. Сви изворни кодови могу се пренети на други рачунар и на њему користити, али сви извршни фајлови и класе морају поново да се компајлирају на том рачунару (на пример класе за Јава програме). Преносиви су само изворни фајлови који се могу читати било којим едитором текста, на пример са *Notepad*.

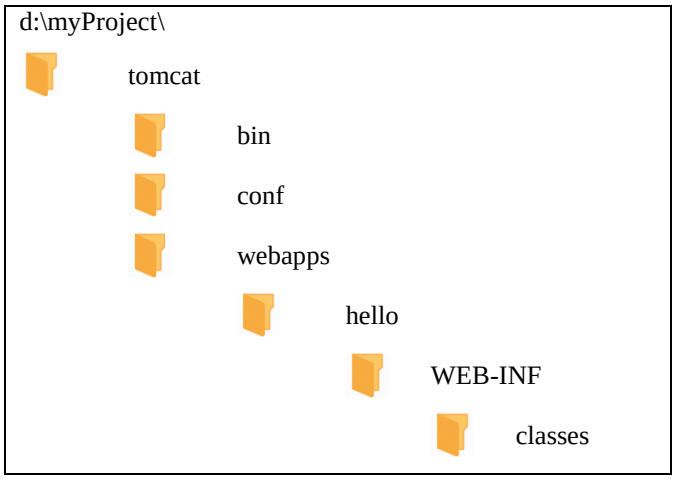
4.1.2 Фолдери *Apache Tomcat* сервера и пример сервлета

У претходним поглављима приказана је инсталација фајлова сервера на клијентском рачунару. У овој теми треба наставити са инсталирањем, што у овом смислу значи да се направе нови фолдери који морају да буду у тачно одређеној структури сервера. Сада свака нова апликација додаје своје нове фолдере, али ми радимо као да настављамо са инсталационим корацима.

4.1.3 Корак 5, систем фолдера и фајлова за први сервлет

Да би направили први сервлет потребно је да направимо нови фолдер у коме ће бити сви фајлови који су у функцији реализације тог сервлета. Претпоставимо да ће назив првог сервлета да буде "hello". Структура фолдера за веб апликацију сервлета "hello" је дата у следећој табели. Знак фасцикле показује да је то фолдер,а затим следи назив фолдера. Све фасцикле које су испод назива фолдера показују да се налазе у том фолдеру.

Табела 4.1.1: Структура директоријума сервера (d:\myProject\ tomcat)



Тако на пример, фолдер "tomcat" је поддиректоријум фолдера "d:\myProject\". Фолдери "bin", "conf" и "webapps" су поддиректоријуми фолдера "tomcat". Фолдер "hello" је поддиректоријум фолдера "webapps". Фолдер "WEB-INF" је поддиректоријум фолдера "webapps". Фолдер "classes" је поддиректоријум фолдера "WEB-INF".

Сервлет "hello" има исто име као фолдер који се налази у фолдеру "webapps". Када клијент у адреси прегледача напише назив сервера, и у наставку назив сервлета (на пример "hello"), то значи да сервер у фолдеру "webapps" тражи фолдер чији је назив исти као назив сервлета ("hello") и у том фолдеру све остале фајлове који су потребни да би сервлет исправно функционисао.

4.1.3.1 Називи фолдера који се произвољно бирају и коју су обавезни

Да би на једном серверу разликовали сервлете, сваки сервлет мора да се налази у истом фолдеру ("webapps") и мора да има другачије називе (у овом случају може да постоји само један фолдер који има назив "hello"). Назив сервлета може произвољно да се бира.

Након избора назива сервлета треба направити фолдер са називом који се налази на диску у нашем случају "d:\myProject\tomcat\webapps\hello". У литератури се често користи скраћени назив "<TOMCAT_HOME>" уместо "d:\myProject\tomcat", зато што програмер може да инсталира сервер у директоријуму који сам изабере, па да не би стално преписивао назив серверског фолдера. У нашем случају би написали да је сервлет у следећем фолдеру "<TOMCAT_HOME>\webapps\hello".

Структура унутар сервлетовог фолдера "hello" је стриктна, и не могу се даље произвољно бирати називи фолдера. Унутар фолдера "hello" налази се фолдер чији је назив "WEB-INF", а у фолдеру "WEB-INF" налази се нови фолдер "classes". Касније ће бити објашњено шта треба да буде у овим фолдерима. Треба обратити пажњу да се у називу "WEB-INF" користе велика слова и обична цртица (минус знак), и да један разлог грешака при развоју

апликација може да буде да се користе и мала слова или такозвана продужена црта, а што апликацију може да збуни као да не постоје ови фолдери. За фолдер "classes" користе се искључиво мала слова, а реч значи множина од речи класа, односно класе.

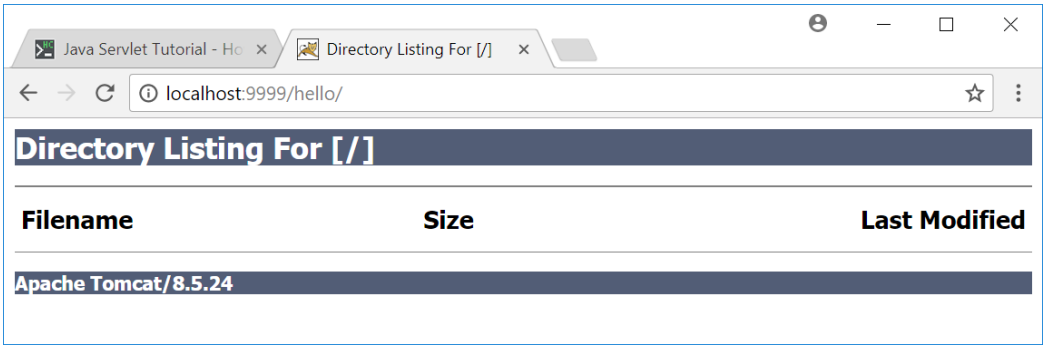
"<TOMCAT_HOME>\webapps\hello" се назива *context root* (или *document base directory*), што би могло да се преведе као основни директоријум веб апликације. Сви *HTML* фајлови и садржаји који су видљиви клијенту (*HTML*, *CSS*, слике, скрипт фајлови, *JSP*) налазе се у овом *context root* директоријуму.

"<TOMCAT_HOME>\webapps\hello\WEB-INF" није видљив клијенту, али је испод *context root* директоријума; овде се чува апликацијски веб дескриптор фајла ("web.xml"). Када би клијент могао да види садржај овог фолдера, он би знао како да приступи сакривеним информацијама, и тада може доћи до озбиљних злоупотреба од стране клијента.

"<TOMCAT_HOME>\webapps\hello\WEB-INF\classes" треба да садржи класе, на пример за прављење сервлет фајлова. Јава програмски језик се користи да се направе фајлови који ће бити одговор на клијентов захтев, а што ће бити детаљно обрађено у наредним темама.

Када је направљен сервлет, треба рестартовати *Tomcat* сервер пре него што се подигне веб апликације; у *Tomcat* конзоли треба да се потврди да је "*hello*" апликација исправно примењена. Разлог за рестартовање сервера је тај да је при стартовању сервера била позната структура, односно садржај директоријума *context root*. Рефрешовање прегледача освежава садржај који је прегледач добио од сервера, и биће другачији приказ ако сервер има нови садржај. Рефрешовање клијентског прегледача нема никакав утицај на сервер, и сервер неће послати нови садржај клијенту ако је дошло до промена у основном директоријуму веб апликације. Због тога, сваки пут када се направи нови фолдер у серверовом директоријуму "webapps" (што значи да је направљен нови сервлет) треба да се рестартује сервер (прекине рад и поново покрене рад сервера).

Иако је у овом примеру направљен само фолдер "hello" у директоријуму "webapps" који је празан, након рестартовања рада сервера, и у прегледачу унесе адреса "http://localhost:9999/hello", прегледач ће приказати садржај директоријума "<TOMCAT_HOME>\webapps\hello" односно d:\myProject\tomcat\webapps\hello", који је још увиек празан, као што се види на слици 4.1.6.



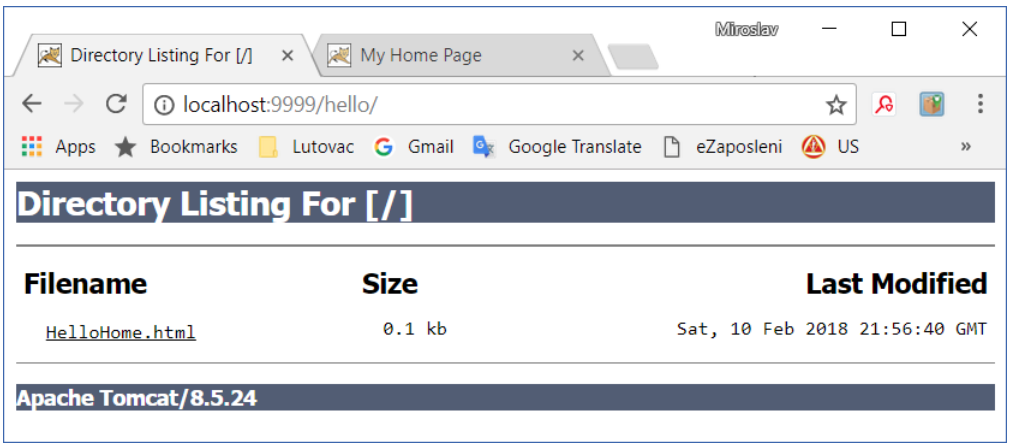
Слика 4.1.6 Приказ прегледача сервлета који је празан

У суштини, направљен је само фолдер где треба да буде сервлет, али пошто у њему нема фајлова и захтеване структуре директоријума, клијентов прегледач приказује садржај празног директоријума на серверу.

Када је наведено да треба да се рестартује сервер, то значу да треба да се кликне у прозор команд промпта (*CMD shell, Command Prompt*) и у њему откуца "shutdown" за прекид рада сервера, а затим да се стартује сервер као што је показано у претходним поглављима тако што се откуца "startup" у одговарајућем фолдеру. Није препоручљиво да се прекид рада уради тако што ће се само затворити *CMD* прозор.

Поновићемо пример из претходног поглавља. Направићемо фајл са називом "HelloHome.html". Пре него што је направљен овај фајл, клијентов прегледач, у који је унето "http://localhost:9999/hello", приказаће садржај директоријума који је празан, као на слици 4.1.7.

Након што је направљен фајл *HelloHome.html*, прегледач у коме је адреса "http://localhost:9999/hello", приказује садржај директоријума који има само један фајл, као на слици 4.1.8.

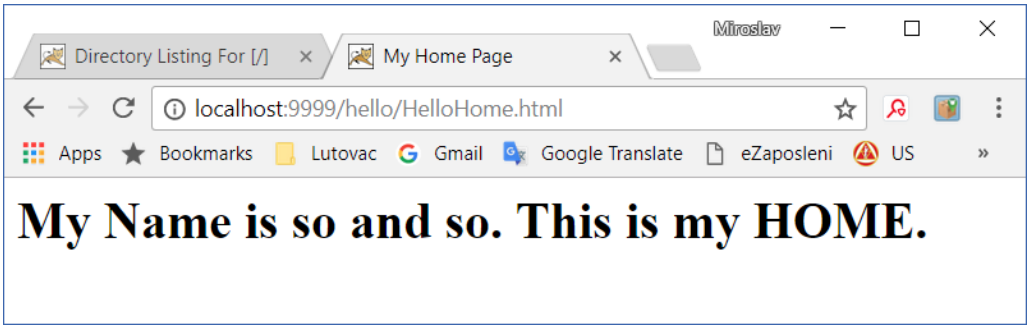


Слика 4.1.7 Приказ прегледача сервлета који има само један фајл

Нека је садржај фајла "HelloHome.html":

```
<html>
  <head><title>My Home Page</title></head>
  <body>
    <h1>My Name is so and so. This is my HOME.</h1>
  </body>
</html>
```

Када клијентском прегледачу додамо *HelloHome.html*, и у прегледачу је адреса "http://localhost:9999/hello/HelloHome.html", приказује се садржај фајла *HelloHome.html*, као на слици 4.1.8.

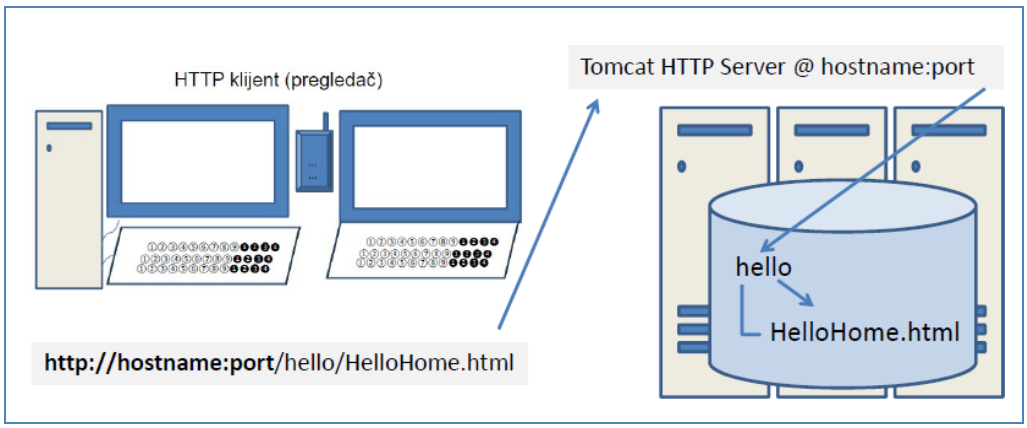


Слика 4.1.8 Приказ *html* фајла

Више приказа прегледача може се истовремено пратити избором језичка у једном прозору прегледача, да не би отворили превише прозора, тако да слике 4.1.7 и 4.1.8 се добијају избором језичка у прозору прегледача.

4.1.3.2 Захтев прегледача и одговор сервера

На слици 4.1.9 је приказано шта се дешава када се на страни клијента у прегледачу у поље адресе унесе адреса сервера у формату *hostname:port*, у наставку назив фолдера (*hello*), а затим и назив фајла *HelloHome.html*.



Слика 4.1.9 Захтев клијента упућен серверу

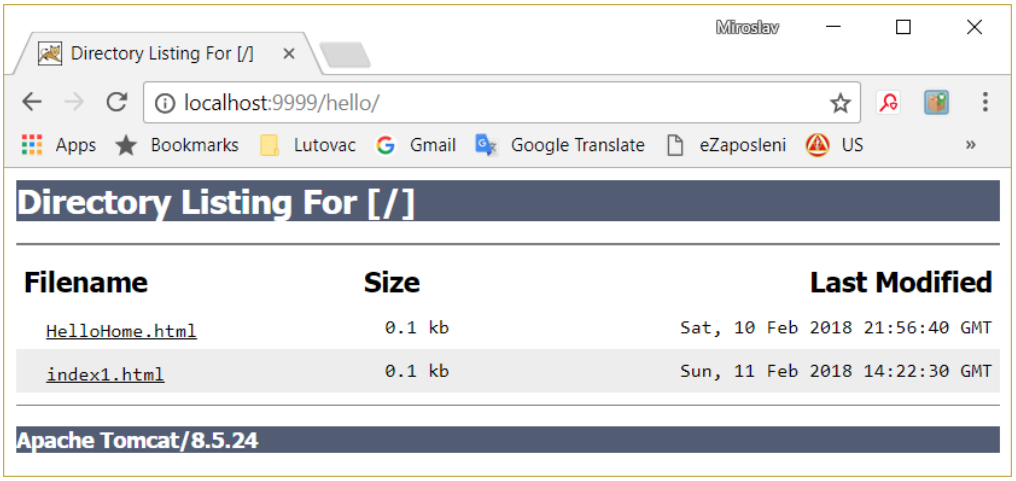
Преко рачунарске мреже (Интернета), а на основу адресе сервера, проналази се путања којом се рутира пренос захтева (поруке) клијентског рачунара. Порука се доставља серверу на адреси *hostname:port*. На страни сервера се проналази директоријум који има назив *hello*. У фолдеру *hello* се тражи фајл *HelloHome.html*.

Ако се не унесе назив фајла (*HelloHome.html*), подразумевани фајл који се тражи је *index.html*. Ако нема фајла *index.html*, приказује се листинг фајлова који се налазе у фолдеру *hello*.

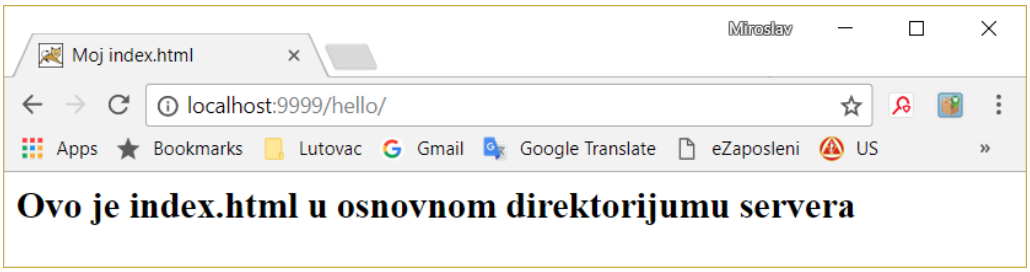
На основу овога можемо да закључимо да је добро да у сваком оваквом фолдеру постоји фајл *index.html*, да клијент не би могао да види садржај фолдера. Овај фајл *index.html* може да садржи опис или нека генерална упутства о томе чему служи овај сервлет, или инструкције шта клијент треба да уради да би добио тражене податке. Алтернативно, при подешавању сервера треба забранити серверу да дозволи клијенту да види садржај фолдера.

На слици 4.1.10 је приказ у прегледачу када се унесе адреса *http://localhost:9999/hello*, и када постоји фајл *index1.html*, али не постоји фајл *index.html*; види се садржај фолдера *hello*.

За исти случај, на слици 4.1.11 је приказ у прегледачу када се унесе адреса *http://localhost:9999/hello*, и када се фајл *index1.html* преименује у фајл *index.html*; Приказује се садржај *index.html*. Треба обратити пажњу да у наслову приказа, у језичку прозора, стоји наслов који објашњава шта се приказује.



Слика 4.1.10 Приказ у прегледачу када нема фајла *index.html* на серверу



Слика 4.1.11 Приказ у прегледачу када постоји фајл *index.html* на серверу

```
D:\myProject\tomcat\bin>ipconfig
Windows IP Configuration
Ethernet adapter Ethernet:
   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
Wireless LAN adapter Local Area Connection* 2:
   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
Ethernet adapter Ethernet 2:
   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
Wireless LAN adapter Wi-Fi:
   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::3901:597d:1942:1dc4%3
   IPv4 Address. . . . . . . . . . . : 192.168.0.15
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.0.1
Tunnel adapter Teredo Tunneling Pseudo-Interface:
   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2001:0:9d38:953c:3c37:27fd:3f57:fff0
   Link-local IPv6 Address . . . . . : fe80::3c37:27fd:3f57:fff0%7
   Default Gateway . . . . . . . . . : ::
D:\myProject\tomcat\bin>
```

Слика 4.1.12 Захтев клијента упућен серверу

Иако све ово може изгледати једноставно, дешава се да нешто не функционише. Ако се појави грешка у приказу прегледача, треба потражити разлог грешке. Ако се добије порука у прегледачу "Unable to Connect", "Internet Explorer cannot display the web page", "404 File Not Found", треба потражити додатне информације у "How to Debug". На пример треба проверити да ли је исправна *IP* адреса.

Стартује се команд промпт (*CMD shell, Command Prompt*) и променом директоријума дође у фолдер "bin" у *Tomcat* инсталационом директоријуму. Када се напише *ipconfig*, добије се приказ као на слици 4.1.12. Како може да се пронађе да ли постоји сервер, описано је у претходним поглављима.

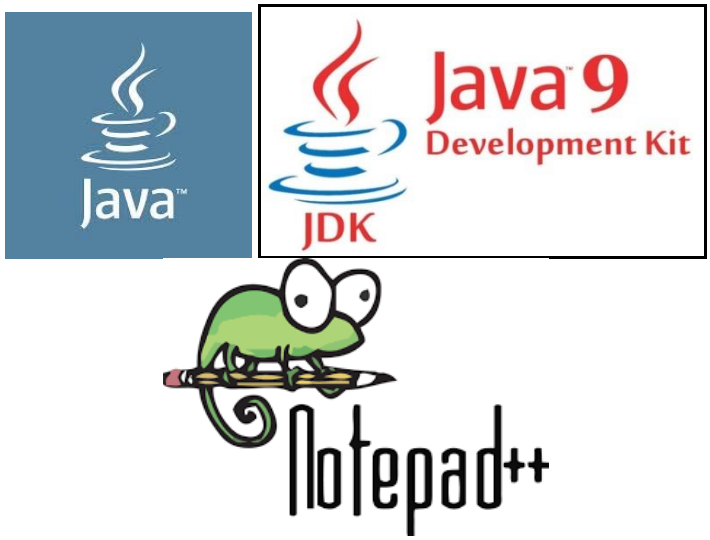
4.1.4 Корак 6., прављење првог сервлета

Ако су сви претходни кораци успешно реализовани, и све је јасно и увежбано, можемо да започнемо најважнији део овог курса, да се направи први једноставан сервлет.

Сервлет је Јава програм који се извршава у оквиру *HTTP* сервера који је базиран на Јави, као што је *Apache Tomcat*. Веб клијент позива сервлет уношењем одговарајуће адресе у веб прегледачу на страни клијента. Да би сво ово могло да се ради потребно је следеће:

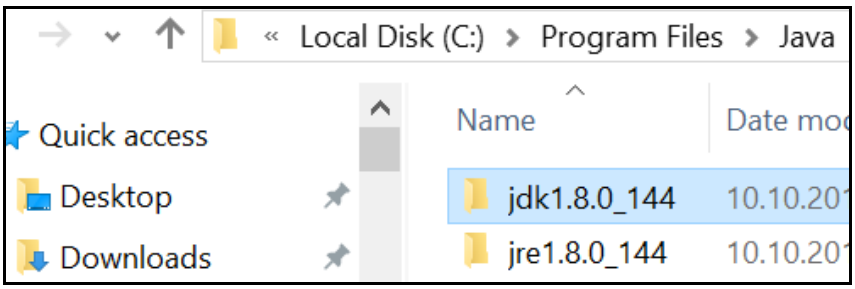
1. Да знате да програмирате у Јави.
2. Да имате инсталиран *JDK* (*Java Development Kit*).
3. Имате на располагању текст едитор као што је *Notepad++* (за *Windows*).

На слици 4.1.13 су приказани графички симболи програма који се користе за сервлете.



Слика 4.1.13 Јава програм, *JDK* (*Java Development Kit*), текст едитор *Notepad++*

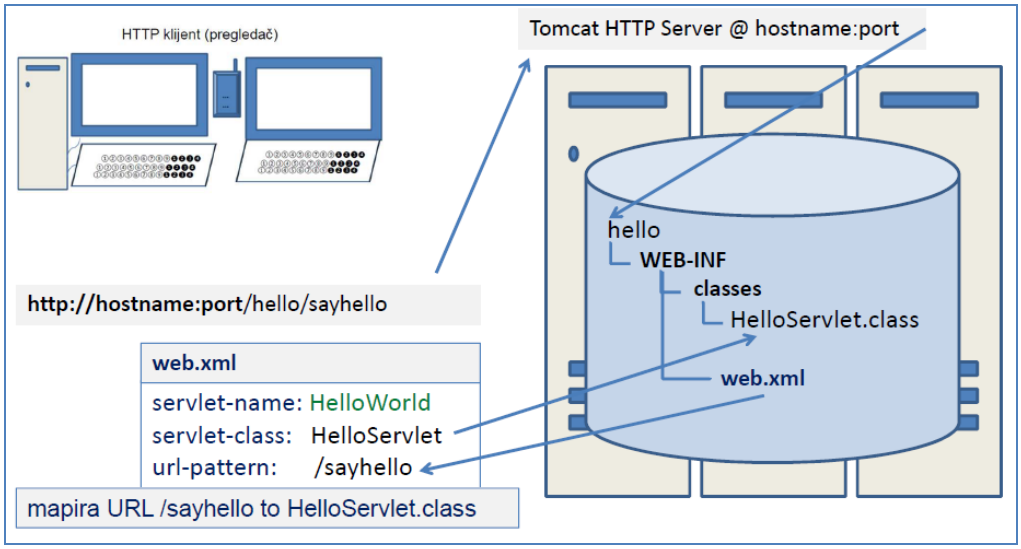
На слици 4.1.14 је приказано како се проверава да ли је инсталиран *JDK* (*Java Development Kit*), и која је верзија.



Слика 4.1.14 Провера верзије која је инсталирана за JDK (*Java Development Kit*)

Јава сервлет је Јава програм који се извршава у оквиру *HTTP* сервера. Корисник на страни клијента позива сервлет исписивањем адресе у прегледачу. У примеру који ће бити описан, треба написати Јава сервлет који се назива *HelloServlet.java*, а који исписује "Hello, world!"; треба написати и конфигурацију сервлета тако да корисник у прегледачу позива овај сервлет коришћењем адресе "http://hostname:port/hello/sayhello".

Произвољно бирамо имена, на пример "hello", "sayhello", односно "hello/sayhello", као и назив фајла "HelloServlet.java", а исписивање "Hello, world!" је према садржају фајла "HelloServlet.java". На слици 4.1.15 је графички приказан ток протока података од клијента до сервера и од сервера до клијента.



Слика 4.1.15 Провера верзије која је инсталирана за JDK (*Java Development Kit*)

На страни клијента у прегледачу се уноси адреса сервера и порт "hostname:port", назив фолдера ("hello") где се тражи сервлет "sayhello",

На страни сервера ("hostname:port") у основном директоријуму се тражи фолдер "hello" у коме је фолдер "WEB-INF" и у њему фајл "web.xml", који преусмерава "sayhello" сервлет на "HelloServlet.class" јава класу.

Називи које програмер може произвољно да бира су "hello", "sayhello", "HelloServlet" (.java и .class), "HelloWorld". Такозване службене речи "WEB-INF", "classes", "web.xml", где је "web.xml" фајл који преусмерава сервлет на јава класу.

На основу адресе унете у прегледач ("sayhello") у основном директоријуму према дефинисаној структури се проналази где је фајл који дефинише где се шта налази и шта треба да се изврши.

4.1.4.1 Пример Јава класе

За писање сервлета потребно је знање јава програмирања; пример "HelloServlet.java" дат је на слици 4.1.16.

```
// To save as "<TOMCAT_HOME>\webapps\hello\WEB-INF\classes\HelloServlet.java"
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloServlet extends HttpServlet {
   @Override
   public void doGet(HttpServletRequest request, HttpServletResponse response)
         throws IOException, ServletException {

      // Set the response MIME type of the response message
      response.setContentType("text/html");
      // Allocate a output writer to write the response message into the network socket
      PrintWriter out = response.getWriter();

      // Write the response message, in an HTML page
      try {
         out.println("<html>");
         out.println("<head><title>Hello, World</title></head>");
         out.println("<body>");
         out.println("<h1>Hello, world!</h1>");  // says Hello
         // Echo client's request information
         out.println("<p>Request URI: " + request.getRequestURI() + "</p>");
         out.println("<p>Protocol: " + request.getProtocol() + "</p>");
         out.println("<p>PathInfo: " + request.getPathInfo() + "</p>");
         out.println("<p>Remote Address: " + request.getRemoteAddr() + "</p>");
         // Generate a random number upon each request
         out.println("<p>A Random Number: <strong>" + Math.random() + "</strong></p>");
         out.println("</body></html>");
      } finally {
         out.close();  // Always close the output writer
      }
   }
}
```

Слика 4.1.16 Пример Јава класе *HelloServlet.java*

Изворни код "HelloServlet.java" треба запамтити у фолдеру

"<TOMCAT_HOME>\webapps\hello\WEB-INF\classes"

Овај сервлет шаље прегледачу клијента *html* фајл који исписује поруку "Hello", ехо неких информација, и случајни број при сваком новом захтеву сервлета.

Садржај "web.xml" ће бити објашњен касније у овој секцији. Сврха сервлета јесте да се на страни сервера генерише *html* фајл који ће обезбедити жељени приказ на страни клијента у прегледачу. Јава програмирање се користи да се динамички мењају садржаји података који се шаљу клијенту.

Да би студенти могли да ископирају текстуални део, следи садржај *HelloServlet.java* фајла:

```
// To save as "<TOMCAT_HOME>\webapps\hello\WEB-INF\classes\HelloServlet.java"
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloServlet extends HttpServlet {
   @Override
```

```
   public void doGet(HttpServletRequest request, HttpServletResponse response)
         throws IOException, ServletException {

      // Set the response MIME type of the response message
      response.setContentType("text/html");
      // Allocate a output writer to write the response message into the
network socket
      PrintWriter out = response.getWriter();

      // Write the response message, in an HTML page
      try {
         out.println("<html>");
         out.println("<head><title>Hello, World</title></head>");
         out.println("<body>");
         out.println("<h1>Hello, world!</h1>");  // says Hello
         // Echo client's request information
         out.println("<p>Request URI: " + request.getRequestURI() + "</p>");
         out.println("<p>Protocol: " + request.getProtocol() + "</p>");
         out.println("<p>PathInfo: " + request.getPathInfo() + "</p>");
         out.println("<p>Remote Address: " + request.getRemoteAddr() +
"</p>");
         // Generate a random number upon each request
         out.println("<p>A Random Number: <strong>" + Math.random() +
"</strong></p>");
         out.println("</body></html>");
      } finally {
         out.close();  // Always close the output writer
      }
   }
}
```

Слика 4.1.17 Јава класа *HelloServlet.java* и приказ у прегледачу

Јава класа се користи да се генерише *html* фајл који ће са сервера бити прослеђен клијенту да га прикаже у прегледачу. Зато се највећи део класе састоји од наредби исписивања линија које ће се појавити у *html* фајлу. У примеру са слике 4.1.17 препознају се тагови за испис наслова у заглављу ("`<head><title>Hello, World</title></head>`", за исписивање текста са предефинисаном величином фонта ("`<h1>Hello, world!</h1>`"), за исписивање текста који је увек исти (на пример "`Request URI: `") у новом параграфу ("`<p> ... </p>`"). Оно што је додатно омогућила јава класа јесте проналажење и исписивање података који нису статични, као на пример "`request.getRequestURI()`". Додатно се ради конкатенација (додавање једном стрингу други стринг коришћењем знака за сабирање, `+`) фиксног непроменљивог текста и динамички добијених података, као на пример "`Request URI: " + request.getRequestURI() ...`". Као илустрација математичких операција, у последњем реду се исписује израчунавање случајног броја који је сваки пут другачији када позовемо сервлет: у загради је шта се исписује, а то је редом, прво се прелази у нови параграф `<p>`, исписује текст "`Random Number: `", наставак текста се исписује већим словима `<strong>`, додаје се тексту вредност случајног броја `+ Math.random()`, затвара се таг за већа слова `</strong>` и затвара се таг за параграф `</p>`. Једино је `" + Math.random() + "` изван знакова наводника (први наводник припада левом делу текста, а последњи наводник припада десном делу текста).

Задатак јава класе јесте да у *html* текст убаци променљиви садржај и да генерише фајл у *html* формату, једини формат који може да разуме сваки прегледач.

4.1.4.2 Компајлирање Јава програма и прављење класе

Јава програми написани као *.java* фајлови могу да се извршавају тек након компајлирања, односно генерисања *.class* фајла.

У примеру који се описује, фајл "HelloServlet.java", треба да се компајлира да би се добио фајл "HelloServlet.class".

Сервлетов *API* (Апликациони програмски интерфејс, енглески назив. *Application Programming Interface*) није део *JDK*; *Tomcat* обезбеђује копију у "<TOMCAT_HOME>/lib/servlet-api.jar". Потребно је укључити овај *JAR* фајл у компајлирање преко *-cp* (*classpath*) опције; *JAR* је *Java ARchive*, фајл формат пакета који се типично користе да агрегирају бројне Јава класе и придруже метаподатке и ресурсе, као што су текст и слике, у фајл за дистрибуцију.

Компајлирање се врши програмом *javac* који се налази у *JDK* фолдеру, и који као изворни фајл користи *.java* фајл, а резултат је *.class* фајл.

Ако постоји *HelloServlet.java* али не и *HelloServlet.class*, појавиће се грешка, приказана на слици 4.1.18. Потребно је компајлирати *.java* програм да би се добио *.class*.

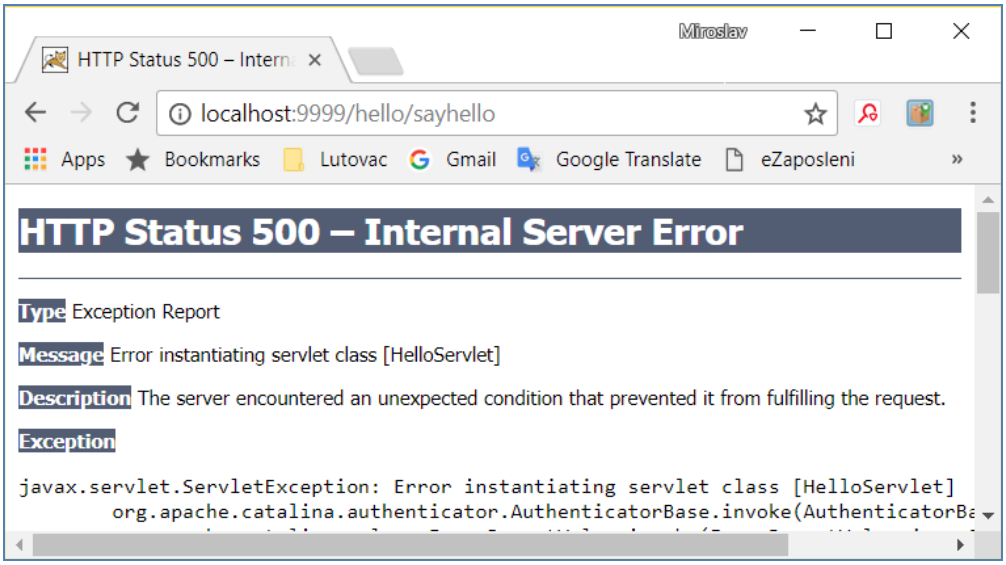
Компајлирање може да се уради у неком интегрисаном развојном окружењу (ИРО) као што су *NetBeans* и *Eclipse*. У овом поглављу биће како да се компајлирање уради без коришћења Интегрисаног развојног окружења (*integrated development environment, IDE*).

Користи се програм *javac* који се налази у *JDK* фолдеру, и који као изворни фајл користи *.java* фајл, а резултат је *.class* фајл.

Потребно је користећи *Command Prompt* извршити низ команди, да би се од *.java* фајла добио *.class* фајл, с тим да оба фајла буду у истом директоријуму *\classes*. Уместо извршавана команди линија по линија, боље је да се напише *.bat* фајл у коме су ове команде. Овај беч (*.bat*) фајл је у */bin* директоријуму; Следи пример за *HelloServlet.java*, а за други сервлет треба променити назив сервлета:

```
:: Podrazumeva se da je Tomcat instaliran u d:\myProject\tomcat
:: Treba promeniti direktorijum u Java osnovni direktorrijum
d:
cd \myProject\tomcat\webapps\hello\WEB-INF\classes
:: ako putanja operativnog sistema ne sadrži put do kompajlera javac
SET PATH=%PATH%;%JAVA_HOME%\bin
:: kompajliranje iz .java u .class
javac -cp .;"d:\myProject\tomcat\lib\servlet-api.jar" HelloServlet.java
```

Уместо да се за коментаре користе две двотачке "`::`" можда је ће за линије које садрже коментар бити потребно да се пише "`//`" или *rem*.



Слика 4.1.18 Грешка се јавља ако постоји *.java* фајл али не и *.class* фајл

Прва линија која није коментар (`d:`) поставља радни директоријум на `D` диск.

Следећа линија поставља радни директоријум у онај где се налази *.java* фајл који треба компајлирати.

```
cd \myProject\tomcat\webapps\hello\WEB-INF\classes
```

Следећа линија треба да постави путању до фолдера "bin" у *Tomcat* инсталационом директоријуму

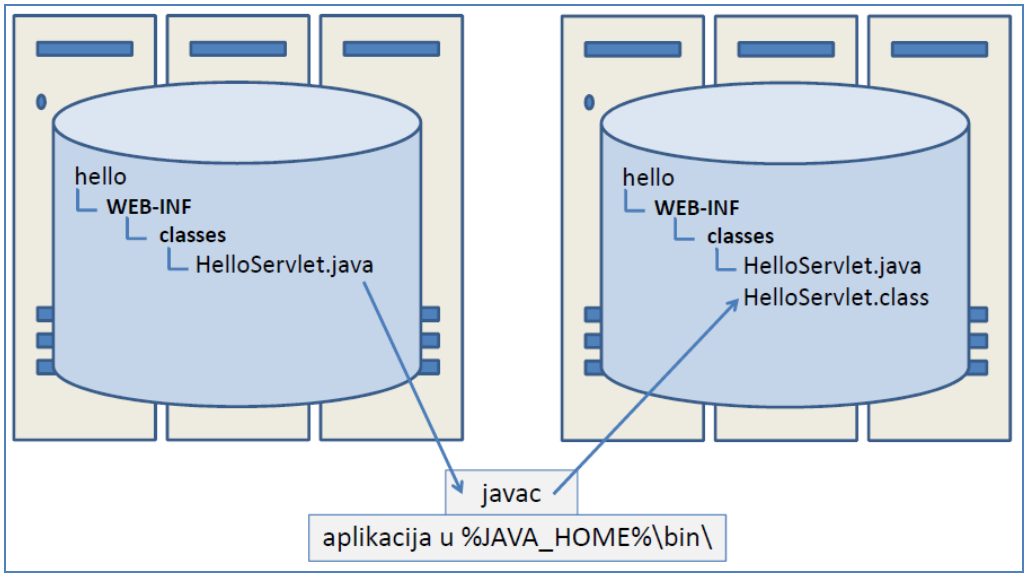
```
SET PATH=%PATH%;%JAVA_HOME%\bin
```

За компајлирање се користи наредна линија која учитава *HelloServlet.java* и генерише компајлирану верзију класе у истом директоријуму (*HelloServlet.class*)

```
javac -cp .;"d:\myProject\tomcat\lib\servlet-api.jar" HelloServlet.java
```

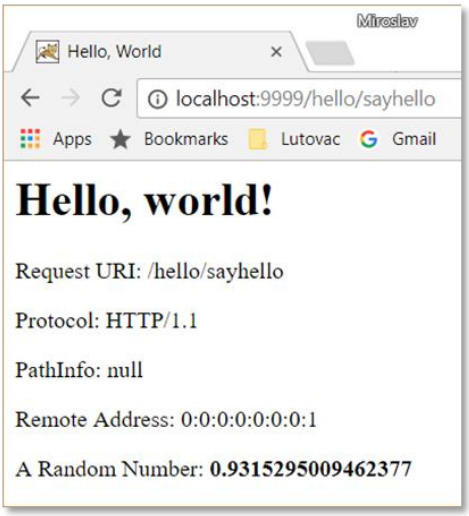
После компајлирања, оба фајла су у фолдеру *classes*.

Графички приказ компајлирања и прављење .class фајл од .java фајла приказан је на слици 4.1.19.



Слика 4.1.19 Графички приказ компајлирања и прављење *.class* фајл од *.java* фајла

Чак и када се уради компајлирање, и направи Јава класа (*HelloServlet.class*), може остати иста грешка у прегледачу. У том случају треба прекинути рад сервера (^C у прозору *command prompt*, а затим и *exit*) и поново стартовати рад сервера као што је раније објашњено.



Слика 4.1.20 Резултат успешног компајлирања и стартовања сервера

Поновним стартовањем сервера, у прегледачу треба да се појави приказ као на слици 4.1.20, с тим да случајни број може да има другачију вредност.

4.1.4.3 Конфигурисање сервлета

За исправан рад потребно је да се конфигурише и сервлет.

Када корисник позове сервлет "sayhello", који се налази на серверу, тако што напише одговарајућу адресу у клијентовом прегледачу, на пример:

"http://hostname:port/hello/sayhello",

потребно је да постоји конфигурациони фајл на адреси сервера који одређује шта се дешава са сервлетом.

У претходном примеру се подразумевало да постоји конфигурациони фајл који има назив "web.xml", и који је смештен у серверовом фолдеру "webapps\hello\WEB-INF"

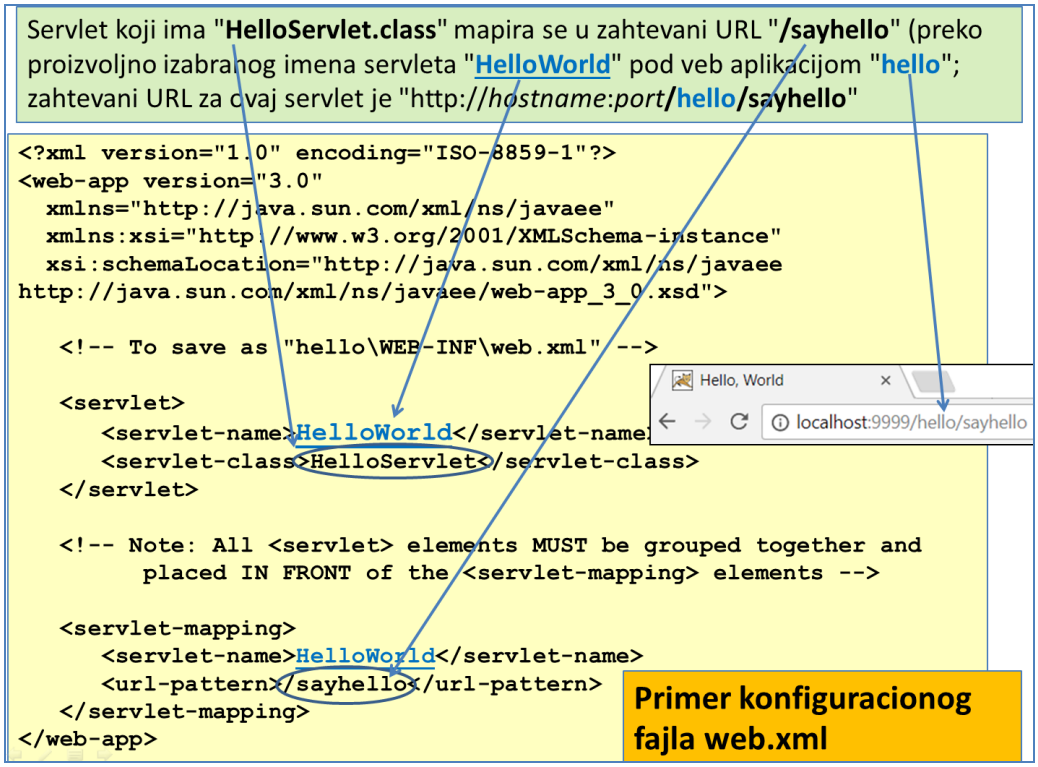
Пун назив са путањом на серверу је следећи

"<TOMCAT_HOME>\webapps\hello\WEB-INF\web.xml"

Речи које морају да буду тачно тако како су написане, које називамо службене речи, и тачно тим редоследом, су

"web.xml"
"webapps\...\WEB-INF"
"... \webapps\...\WEB-INF\web.xml"

Речи које програмер може произвољно да бира су назив фолдера "hello", и назив сервлета "sayhello". Пример конфигурационог фајла са објашњењима је дат на слици 4.1.21.



Слика 4.1.21 Пример конфигурационог фајла

Последња реч у адреси коју прозива клијентов прегледач, у ствари је назив сервлета. Сервер у конфигурационом фајлу тражи ту реч између тагова "<servlet-mapping>" и "</servlet-mapping>", као реч која се појављује између тагова "<url-pattern>" и "</url-pattern>" као "/sayhello".

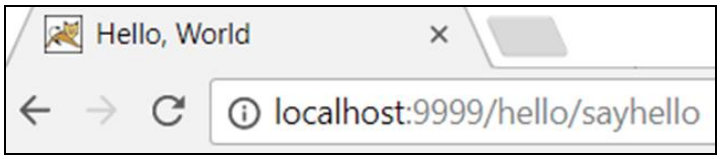
Сервер у конфигурационом фајлу има у пару између тагова "<servlet-mapping>" и "</servlet-mapping>", још један пар тагова, "<servlet-name>" и "</servlet-name>" између којих је произвољно изабрана реч "HelloWorld".

Иста ова реч "HelloWorld" се појављује између тагова "<servlet-name>" и "</servlet-name>", који су унутар тагова "<servlet>" и "</servlet>". Између тагова "<servlet >" и "</servlet >" налази се још један пар тагова <servlet-class> и </servlet-class> између којих је назив "HelloServlet", и то је назив класе (*HelloServlet.class*) који се користи за генерисање *html* фајла који ће бити прослеђен клијентовом прегледачу.

Да сада поновимо цео ток дешавања.

Када клијент у свом прегледачу унесе адресу (у нашем случају према слици 22 то је "localhost:9999/hello/sayhello"), адреса сервера је "localhost", порт на серверу је ":9999",

фолдер где се налази сервлет је "/hello" (и налази се у директоријуму "webapps"), сервер и фолдеру ""hello/WEB-INF" проналази фајл "web.xml".

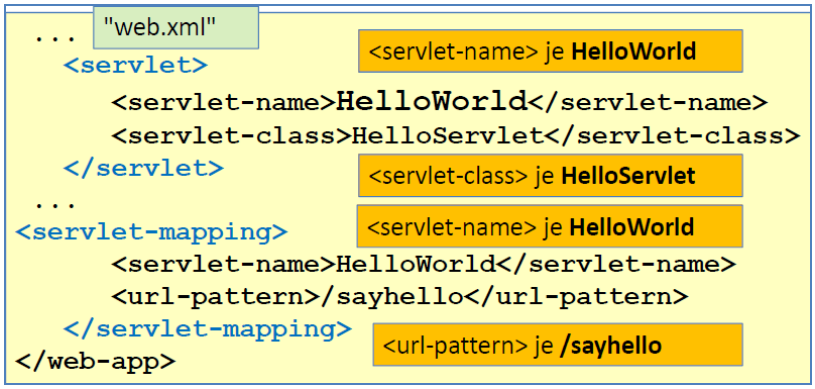


Слика 4.1.22 Пример конфигурационог фајла

Сервер проналази у фајлу "web.xml" реч из адресе "sayhello" између тагова <servlet-mapping>"..."<url-pattern>" "sayhello" "</url-pattern>"..."</servlet-mapping>" и упарује је са називом који је између тагова где проналази име "HelloServlet", у овом случају то је
<servlet-mapping>"..."<servlet-name>" "HelloWorld" "</servlet-name>"..."</servlet-mapping>"
и

Ово име се користи за мапирање на класу тако што се прво проналази исто име између тагова <servlet>"..."<servlet-name>" "HelloWorld" "</servlet-name>"..."</servlet>" и име класе које је између тагова
<servlet>"..."<servlet-class>" "HelloServlet" "</ servlet-class>"..."</servlet >"

На слици 4.1.23 су посебно означени делови фајла "web.xml" где се види мапирање назива сервлета "sayhello", из адресе унете у клијентов прегледач, у произвољно изабрану реч "HelloWorld", а преко исте речи "HelloWorld" у назив класе (*HelloServlet.class*). Фајл "web.xml" се налази у фолдеру "WEB-INF", а фајл "HelloServlet.class" се налази у фолдеру "classes". Фајл "HelloServlet.class" се извршава и он генерише *html* фајл који се шаље клијентовом прегледачу, као што је илустровано на слици 4.1.20.



Слика 4.1.23 Део конфигурационог фајла за мапирање адресе у Јава класу

Структура директоријума сервера за овај пример је приказана у Табели 4.1.1.

Да би студенти могли да копирају конфигурациони фајл, следи садржај *web.xml*:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="3.0"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

   <!-- To save as "hello\WEB-INF\web.xml" -->

   <servlet>
      <servlet-name>HelloWorld</servlet-name>
      <servlet-class>HelloServlet</servlet-class>
   </servlet>

   <!-- Note: All <servlet> elements MUST be grouped together and
         placed IN FRONT of the <servlet-mapping> elements -->

   <servlet-mapping>
      <servlet-name>HelloWorld</servlet-name>
      <url-pattern>/sayhello</url-pattern>
   </servlet-mapping>
</web-app>
```

4.1.5 Конфигурисање већег броја сервлета

Сви сервлети се налазе у унутар фолдера "webapps". Ако се прави веб апликација *hello*, тада се унутар фолдера "webapps" налази фолдер "hello". Конфигурациони фајл *web.xml* важи само за веб апликацију "hello" и он је у фолдеру "hello\WEB-INF" односно он се налази у "webapps\hello\WEB-INF\web.xml".

Неки други сервлет мора да има другачији назив, на пример "xxx", и он се налази унутар фолдера "webapps", и има увек исту структуру фолдера "WEB-INF" и "classes", и свој конфигурациони фајл web.xml који је написан за конфигурацију сервлета "xxx", "webapps\xxx\WEB-INF\web.xml". Другим речима, сваки сервлет има свој конфигурациони фајл, а до забуне не може да дође зато што се сваки од њих налази у фолдеру који одговара називу сервлета.

Након прављења или измена у конфигурационом фајлу, потребно је да се рестартује *Tomcat* сервер (да се стопира рад сервера, и да се поново стартује сервер) да би се освежио важећи конфигурациони фајл.

Важно је напоменути да сваки сервлет мора да се појављује у паровима како би мапирање било без грешака. Сви "<servlet>" елементи морају бити заједно груписани и смештени испред "<servlet-mapping>" елемената.

4.1.6 Позивање сервлета

Када је направљена Јава класа, добијен компајлирани фајл, конфигурисан сервлет, може се позвати сервлет са клијентовог прегледача уношењем адресе, на пример

"http://hostname:port/hello/sayhello"

На слици 4.1.24 су приказани типични прикази. Ако није стартован сервер, појавиће се грешка. Ако сервер не постоји или није добро конфигурисан, појавиће се грешка. Ако је стартован сервер и све коректно урађено, појавиће се то што ради јава класа, у овом примеру на слици у средини. Када се кликне дугме за поновно учитавање, све је исто осим што је добијен други случајни број.

Слика 4.1.24 Примери рада сервлета.

Након што је позван сервлет на серверу, са сервера је послат *html* фајл који се приказује у прегледачу клијента.

Веб прегледач клијента прима само излазни текст који је генерисао сервлет, а где су линије текста резултати "out.println()" команди јава класе. Клијент нема приступ сервлетовом изворном програму који може садржати поверљиве информације које сервер не жели да клијент зна.

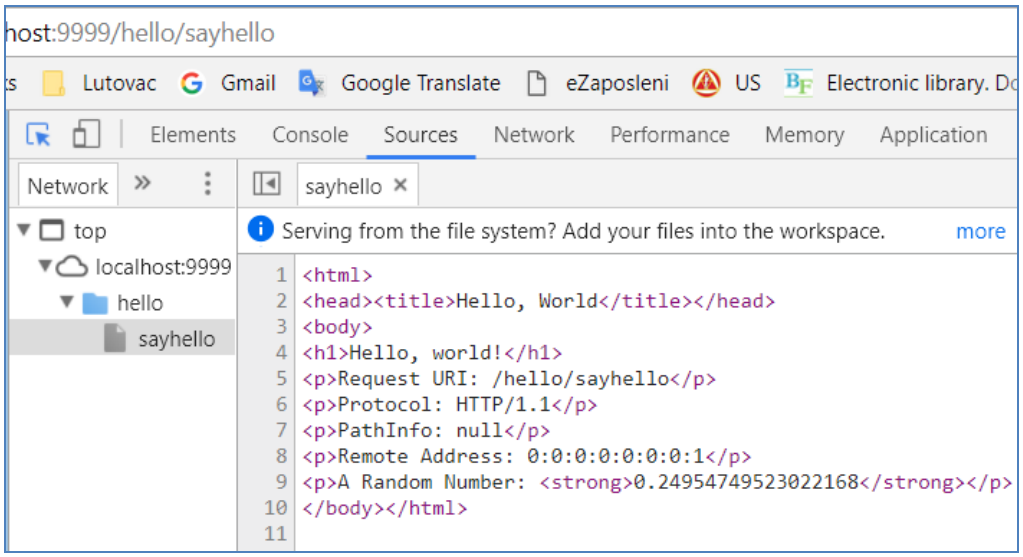
Пример *html* фајла који добија прегледал дат је на слици 4.1.24 у средини, а текстуелни део би био следећи

```
<html>
<head><title>Hello, World</title></head>
<body>
<h1>Hello, world!</h1>
<p>Request URI: /hello/sayhello</p>
<p>Protocol: HTTP/1.1</p>
<p>PathInfo: null</p>
<p>Remote Address: 127.0.0.1</p>
<p>A Random Number: <strong>0.6465595957301623</strong></p>
</body>
</html>
```

При поновном позивању, добила би се на пример десни део слике 4.1.24, а у претходном текстуалном делу *html* фајла би био другачији случајни број.

4.1.6.1 Детаљи о комуникацији клијента и сервера

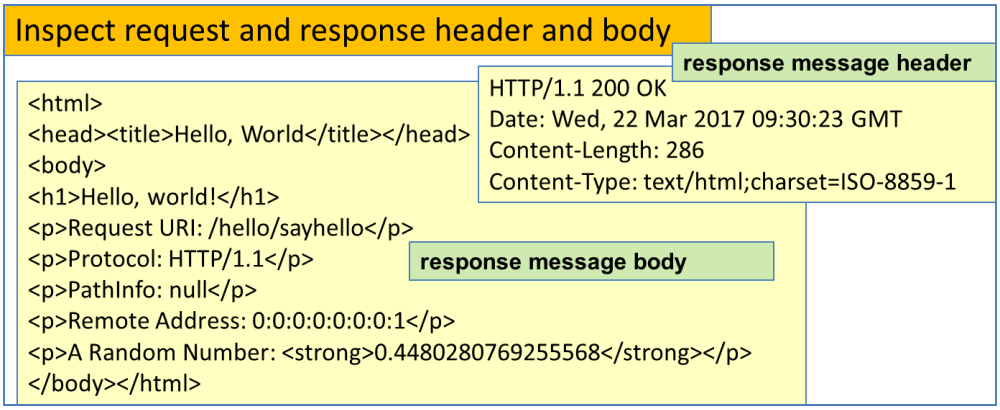
У прегледачу *Chrome*, кликне се на *F12* да би се омогућило коришћење "developer tool". Изабрати "Network" палета и унесе *URL* адреса. Добија се приказ као на сликама 4.1.25 – 4.1.28.



Слика 4.1.25 Приказ коришћења *developer tool*

Слика 4.1.26 Приказ коришћења *developer tool*

Слика 4.1.27 Приказ коришћења *developer tool*



Слика 4.1.28 Приказ коришћења *developer tool*

Неки од интересантних детаља се могу извући за даљу анализу:

```
Accept: text/html,application/xhtml+xml...
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cache-Control: max-age=0
Connection: keep-alive
Host: localhost:9999
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Win64; x64)
```

На пример, прегледач садржи и податак о томе да ли је рачунар 32-битни или 64-битни.

4.1.7 Питања и задаци за вежбу

1. Направити једноставан сервлет који ће поновити пример из ове теме, с тим да се приказује другачији текст у прегледачу и користе другачији називи за сервлет и Јава класу.
2. Написати извештај са описом евентуалних проблема.

4.2 Сервлети са *HTML* формом

Резиме

Циљ овог поглавља је да студент разуме функционисање сервлета израдом једноставних примера. Дати су детаљни поступци од постављања задатка, припреме софтверског окружења и тестирање и верификацију софтверског решења.

Увод у сервлете и HTML форме

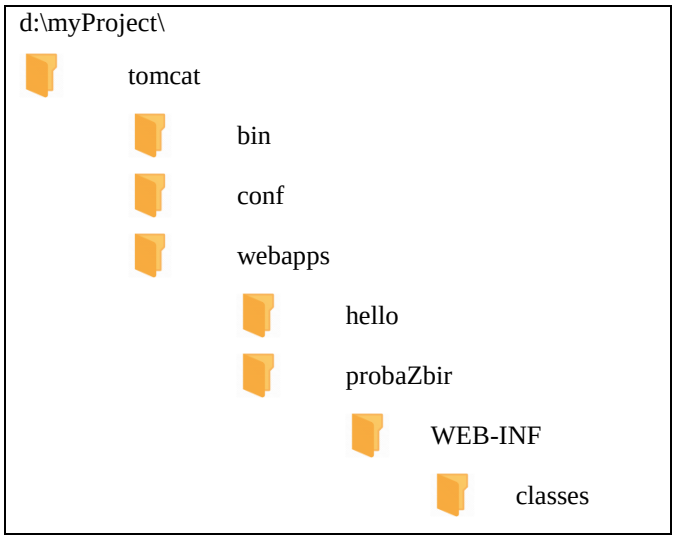
У овом поглављу је дат детаљан опис инсталације сервлета са једним једноставним примером у коме је коришћена једна Јава класа да се генерише *HTML* фајл који сервер шаље клијенту када од клијента стигне захтев за подацима.

У овој теми ће бити обрађен незнатно сложенији задатак како би студенти поступно могли да овладају овом материјом. У овој теми се треба реализовати сервлет који је решење за Задатак 1 из Програмирање интернет апликација, уџбеник са збирком задатака, Б. Николић и Д. Драшковић: Коришћењем Јава Сервлета написати сервлет који врши сабирање два броја, и приказује збир у прегледачу. Унос података треба извршити кроз *HTML* форму.

4.2.1 Избор назива фолдера и фајлова

Да би направили сервлет, који се налази на серверу, потребно је да се изаберу називи сервлета, фолдера и фајлова, направе одговарајући фолдери на серверу и напишу фајлови.

Табела 4.2.1. Структура директоријума сервера (d:\myProject\ tomcat)



Прво треба да се изабере назив фолдера за сервлет у коме ће се налазити сви фајлови за овај задатак. Изаберимо име које нас подсећа на то шта сервлет треба да уради: *probaZbir*

је назив фолдер који треба да се налази унутар фолдера *webapps*; у претходном делу то је био фолдер *hello*, који и даље остаје као поддиректоријум фолдера *webapps*, а сада се додаје нови фолдер *probaZbir*, у новом фолдеру треба да постоје фолдери *WEB-INF* и *classes*, као што је приказано у Табели 4.2.1.

Фолдер *hello* има своје фолдере *WEB-INF* и *classes*, и они нису приказани у Табели 4.2.1.

Некада је добро направити копију целог фолдера *hello*, а затим копију преименовати да има другачији назив, *probaZbir*. У том случају нећемо заборавити да направим неке од неопходних фолдера, при чему треба променити називе фолдерима и фајловима да би нзиви одговарали функцији која се остварује.

4.2.2 Прављење *html* фајла за позив сервлета

У првом примеру који је реализован у оквиру поглавља о сервлетима, клијент уноси адресу сервера, и одмах у наставку уноси назив фолдера где се налази сервлет и назив који се користи за мапирање фајлова који се налазе у конфигурационом фајлу *web.xml*, како би пронашао Јава класу која извршава жељени задатак (у овом тренутку још увек није направљена класа, али смо изабрали назив класа *probaServletZbir.class*).

Како у овом задатку треба да омогућимо да клијент прво унесе неке податке (два броја која треба да буду сабрана), па тек онда да Јава класа изврши операцију сабирања (*probaServletZbir.class*), то не треба одмах позивати сервлет. Клијент може да користи *html* фајл за унос бројева, па тек када коректно унесе бројеве може да позива класу која сабира унете бројеве. Фајл који најпре треба да види клијент је у *html* формату, и он служи да се унесу бројеви, да се провери да ли су бројеви коректно унети, па тек након тога из *html* фајла треба позвати сервлет да изврши своју функцију (да сабере бројеве).

Произвољно бирамо назив фајла, на пример *probaZbirHTML.html*, а може да буде и *index.html*. Ови *html* фајлови треба да се налазе у већ направљеном фолдеру *probaZbir*.

Уместо уношења адресе "http://localhost:9999/probaZbir/", може да се дода и назив фајла "http://localhost:9999/probaZbir/probaZbirHTML.html". Обзиром да оба фајла имају идентичан садржај, "index.html" и "probaZbirHTML.html", и добија се исти приказ у прозору прегледача као на слици 4.1.1. Са становишта клијента, боље је да се користи "index.html", зато што ако се погреши у називу *html* фајла, биће приказан садржај фолдера "probaZbir", што често не желимо да види клијент.

У примеру који се прави, овај *html* фајл треба да садржи форму за унос два броја која сервлет треба да сабере.

На сервер страни направљена је структура директоријума и сада треба да направимо *html* фајл. Касније, биће направљени *xml* и *class* фајлови.

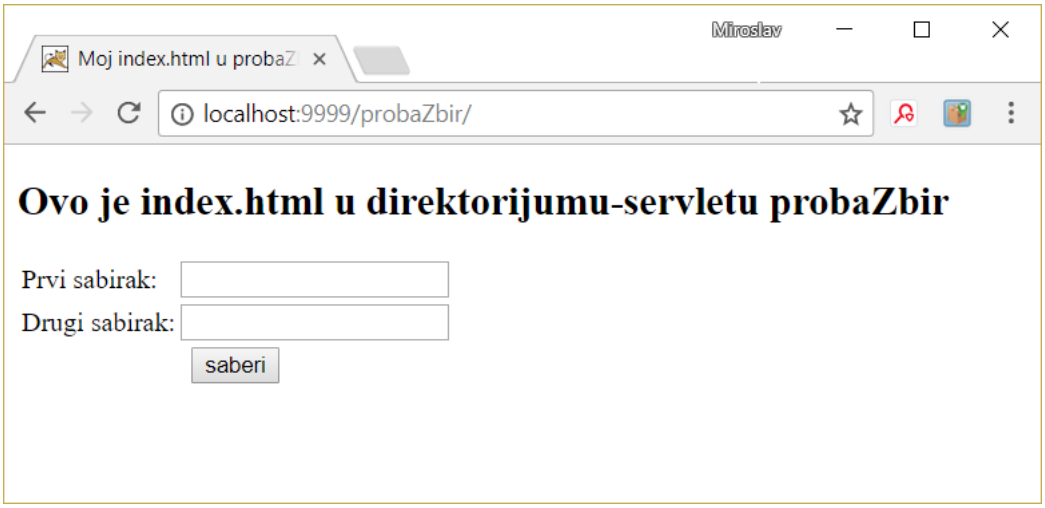
Садржај *html* фајла у који треба да се унесу бројеви је *probaZbirHTML.html*:

```
<!DOCTYPE html>
<html>
  <head><title>Moj probaZbirHTML</title>
       <meta http-equiv="Content-Type" content="text/html;
           charset=UTF-8">
  </head>
```

```
        <body>
          <h2>Ovo je probaZbirHTML u direktorijumu-servletu probaZbir</h2>
             <form action="saberiBrojeve" method="POST">
                <table>
                  <tr>
                    <td>Prvi sabirak:</td>
                    <td><input type="text" name="prvi" value=""/> </td>
                  </tr>
                  <tr>
                    <td>Drugi sabirak:</td>
                    <td><input type="text" name="drugi" value=""/> </td>
                  </tr>
                  <tr>
                    <td colspan="2" align="center">
                    <input type="submit" value="saberi" />
                    </td>
                  </tr>
                </table>
             </form>
      </body>
      </html>
```

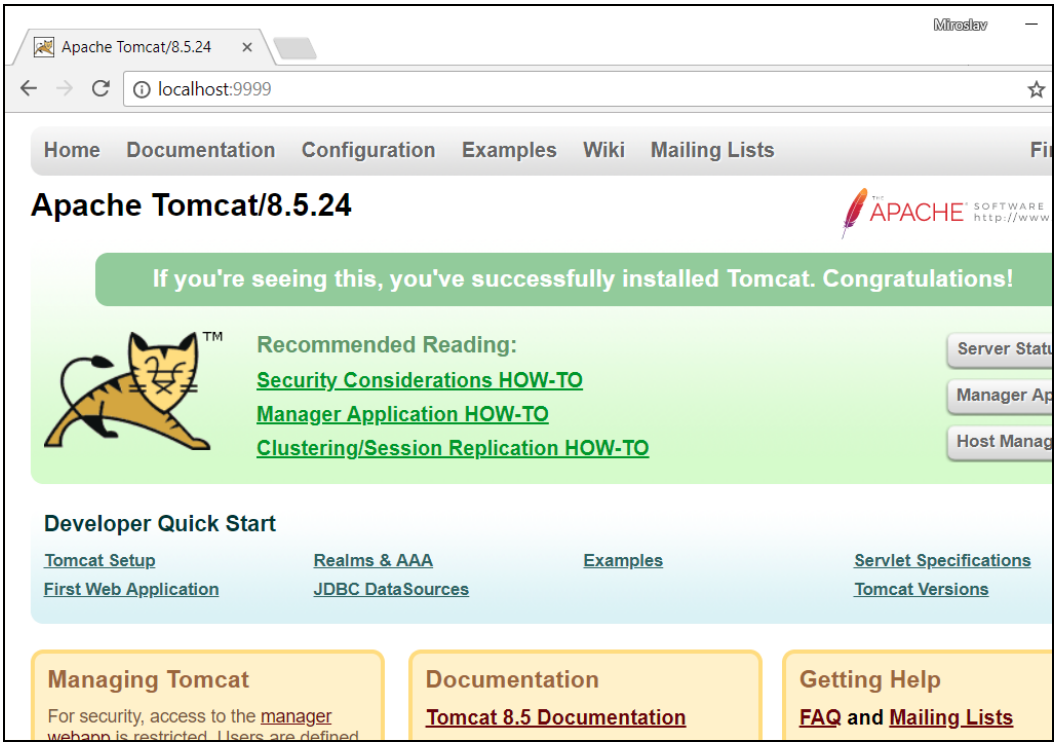
Фајл *probaZbirHTML.html* можемо да ископирамо и под другим називом, *index.html*.

Сада можемо да урадимо први део задатка, да стартујемо сервер, да стартујемо клијентов прегледач који шаље захтев серверу "http://localhost:9999/probaZbir/", уношењем адресе сервера "http://localhost", број порта на серверу ":9999", а у наставку уносимо и назив фолдера у коме очекујемо да се изврши сервлет "/probaZbir/". Сервер проналази фајл *index.html*, и коначно приказа у клијентовом прегледачу дата је на слици 4.2.1.



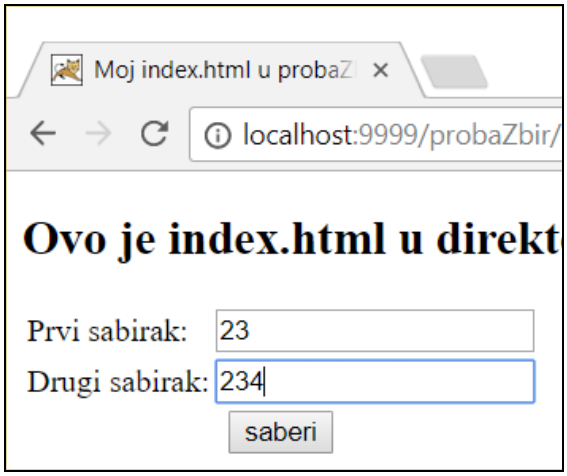
Слика 4.2.1 Одзив сервера на захтев клијента на основу фајла *index.html*

Уколико се не добије приказани садржај, треба проверити да ли је стартован сервер. То може да се провери ако се унесе у прегледачу само први део адресе, на пример, "http://localhost:9999/", уношењем адресе сервера "http://localhost" и порта ":9999", када се добија приказ са слике 4.2.2 са информацијама о серверу.



Слика 4.2.2 Провера да ли је сервер стартован

Након уношења два броја, у форму са слике 4.2.1, добија се приказ прегледача као на слици 4.2.3.



Слика 4.2.3 Приказ прегледача када се унесу два броја

Када погледамо *html* код, и део у коду

```
<form action="saberiBrojeve" method="POST">
   <table>
       ...
     <tr>
       <td colspan="2" align="center">
       <input type="submit" value="saberi" />
       </td>
     </tr>
   </table>
</form>
```

Препознајемо да се овим кодом прави дугме у *html* форми, и да на дугмету пише "saberi", што нас асоцира на то треба да кликнемо на дугме и да тада очекујемо да унети бројеви буду сабрани.

У почетку овог сегмента кода постоји линија кода

```
<form action="saberiBrojeve" method="POST">
```

Овај део кода показује шта треба да се догоди када се кликне на дугме "saberi". Метод "`POST`" који је изабран додаје реч "`saberiBrojeve`" на адресу прегледача. То је исто као да смо написали у прегледачу следећу адресу:

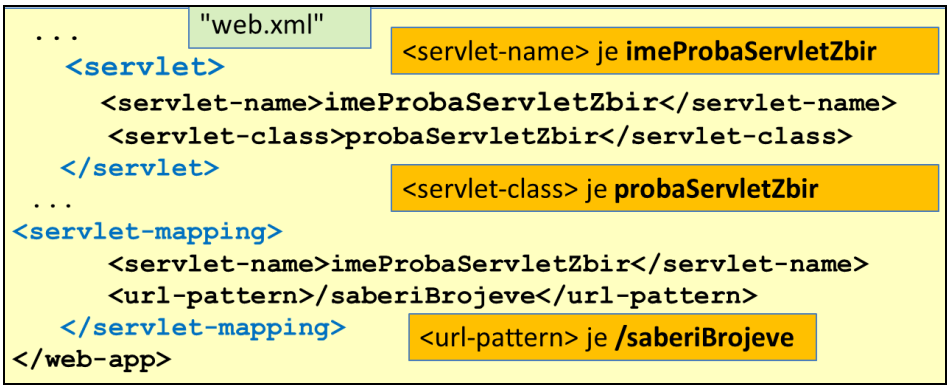
"http://localhost:9999/probaZbir/saberiBrojeve"

Оваква адреса нас подсећа на пример из претходне теме, где је http://localhost:9999 адреса сервера, "probaZbir" је назив фолдера на серверу, а "saberiBrojeve" реч која се користи у конфигурационом фајлу за дефинисање мапирања фајлова.

4.2.3 Прављење конфигурационог фајла за сервлет

Поддиректоријум фолдера "webapps" је директоријум "probaZbir", а поддиректоријум директоријум "probaZbir" је директоријум "WEB-INF", У фолдеру "WEB-INF" налази се конфигурациони фајл "web.xml". Овај "web.xml" фајл важи само за сервлет "probaZbir".

На слици 4.2.4 је приказан део "web.xml" где се види како се остварује мапирање.



Слика 4.2.4 Део конфигурационог фајла

Из адресе у прегледачу "http://localhost:9999/probaZbir/saberiBrojeve", сервер проналази у конфигурационом фајлу "web.xml" последњу реч "saberiBrojeve", и препознаје која се реч користи за мапирање, "imeProbaServletZbir". У линијама које претходе се појављује иста ова реч "imeProbaServletZbir", а као пар са овом речи је назив класе "probaServletZbir". Ово значи да ће се последња реч из адресе у прегледачу "saberiBrojeve" мапирати у класу "probaServletZbir", односно биће покренут фајл "probaServletZbir.class". Задатак класе "probaServletZbir.class" је да прихвати два броја унета у *html* форму, да их сабере и пошаље прегледачу нови *html* фајл, тако да прегледач може да прикаже резултат сабирања.

Ради комплетности, дат је садржај конфигурационог фајла:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="3.0"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

   <!-- To save as "...\WEB-INF\web.xml" -->

   <servlet>
      <servlet-name>imeProbaServletZbir</servlet-name>
      <servlet-class>probaServletZbir</servlet-class>
   </servlet>

   <!-- Note: All <servlet> elements MUST be grouped together and
```

```
                placed IN FRONT of the <servlet-mapping> elements -->

          <servlet-mapping>
             <servlet-name>imeProbaServletZbir</servlet-name>
             <url-pattern>/saberiBrojeve</url-pattern>
          </servlet-mapping>
       </web-app>
```

4.2.4 Прављење Јава класе

Из "web.xml" знамо да се сервлет мапира у класу "probaServletZbir.class". За писање класе потребно је знање Јава програмирања. Прво треба направити Јава програма, "probaServletZbir.java". Задатак програма је да прихвати унете бројеве из *index.html* фајла и да израчуна збир, а да затим генерише *html* фајл који се шаље прегледачу. Следи код за "probaServletZbir.java" који се налази у фолдеру "classes":

```
//source file: probaServletZbir.java

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class probaServletZbir extends HttpServlet {
 //metoda za obradu i HTTP GET i HTTP POST zahteva
   protected void processRequest(HttpServletRequest request,
                                 HttpServletResponse response)
   throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
         PrintWriter out = response.getWriter();
         String prvi=request.getParameter("prvi");
         String drugi=request.getParameter("drugi");
         int a=0, b=0;
         boolean ispravno=true;
         //double c=0.0;

         try{
                a=Integer.parseInt(prvi);
                b=Integer.parseInt(drugi);
                //c = Double.parseDouble(prvi);
         }catch(NumberFormatException nfe){ispravno=false;}

   int zbir=a+b;

   try {
          out.println("<html>");
          out.println("<head>");
          out.println("<title>Primer2</title>");
          out.println("</head>");
          if(ispravno){
                 out.println("<body bgcolor=\"00ffff\">");
                 out.println("<h1>Zbir je: " + zbir + "</h1>");
```

```
            }
            else{
                   out.println("<h1>Neispravni brojevi!</h1>");
            }
            out.println("</body>");
            out.println("</html>");
        } finally {
              out.close();
          }
    }

protected void doGet(HttpServletRequest request,
                     HttpServletResponse response)
throws ServletException, IOException {
   processRequest(request, response);
}
protected void doPost(HttpServletRequest request,
                      HttpServletResponse response)
throws ServletException, IOException {
                      processRequest(request, response);
}
public String getServletInfo() {
        return "Short description";
    }
}
```

Програми са екстензијом јава не могу се извршавати. Прво треба да буду компајлирани.

4.2.5 Компајлирање Јава програма и прављење класе

Јава програми написани као *.java* фајлови могу да се извршавају тек након компајлирања, односно генерисања *.class* фајла.

У примеру који се описује, фајл "probaServletZbir.java", треба да се компајлира да би се добио фајл "probaServletZbir.class". За компајлирање се користи програм javac.

За компајлирање се може направити .bat фајл, у коме су одговарајуће команде, као што је објашњено у претходној теми. Ако је компајлирање протекло без грешака, добиће се одговарајућа класа. Ако има грешака, може се користити неко од окружења за развој Јава апликација које омогућавају лакше проналажење грешака у коду.

4.2.6 Тестирање рада апликације

За тестирање апликације, покрене се сервер и у прегледачу се унесе адреса, "http://localhost:9999/probaZbir/". Сервер проналази фајл "index.html", интерпретира га и даје приказ као на слици 4.2.1. Након уношења бројева које треба сабрати, добија се слика 4.2.3. Кликне се на дугме сабери. Прегледачу се у адреси након клика на дугме додаје реч "/saberiBrojeve", па је адреса "http://localhost:9999/probaZbir/saberiBrojeve". Сервер у фолдеру апликације у конфигурационом фајлу проналази у коју се класу мапира сервлет и извршава класу. Класа сабира бројеве и шаље *html* фајл прегледачу онако како га је направио "probaServletZbir.class", као што је приказано на слици 4.2.5.

Слика 4.2.5 Одзив сервлета на као збир два броја

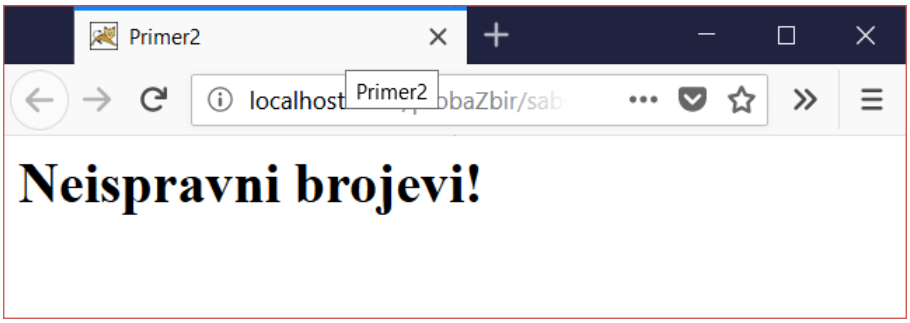
Код који је примио клијентов прегледач је следећи:

```
<html>
   <head>
      <title>Primer2</title>
   </head>
   <body bgcolor="00ffff">
      <h1>Zbir je: 257</h1>
   </body>
</html>
```

Важно је уочити да у адреси прегледача и даље стоји иста адреса,

"http://localhost:9999/probaZbir/saberiBrojeve"

Клијент не зна која класа је направила овај код и овакав приказ. Када би клијент одмах унео "http://localhost:9999/probaZbir/saberiBrojeve", добио би слику 4.2.6.



Слика 4.2.6 Одзив сервлета без уноса два броја за сабирање

Код директног позивања класе, нема *html* форме за унос бројева, па јава класа не зна који су бројеви унети а које треба да сабере и зато јавља грешку.

Јава класа је написана тако да у овом случају генерише другачији *html* код:

```
<html>
  <head>
     <title>Primer2</title>
  </head>
  <body>
      <h1>Neispravni brojevi!</h1>
  </body>
</html>
```

Сада су јаснији разлози за примену сервлета. Потребно је од клијената сакрити како се и шта ради, како се мапирају фајлови, и какав ће све исход комуникације да буде, и истовремено онемогућити да се клијент укључи у средину комуникације без претходно пренетих и сакривених података.

У случају да се уместо "index.html" користи фајл "probaZbirHTML.html ", добија се исти одговор и сервер се понаша на исти начин.

4.2.7 Питања и задаци за вежбу

1. Направити сервлет који ће поновити пример из ове теме, с тим да се у *html* форми уноси име и презиме, а Јава програм враћа поздравну реченицу у којој користи име и презиме.
2. Написати извештај са описом евентуалних проблема.

4.3 Рад са више сервлета

Резиме

Циљ овог поглавља је да се студент оспособи за израду сложенијих сервлета.

Увод у

У претходном поглављу у дата два детаљна описа примера за реализацију сервлета.

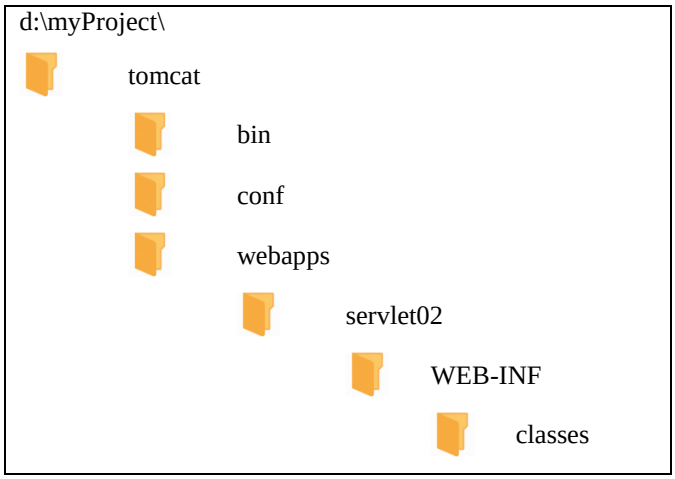
У овом поглављу ће бити трећи пример са сложенијим задатком.

У овој теми се треба реализовати сервлет који је решење за Задатак 2 из Програмирање интернет апликација, уџбеник са збирком задатака, Б. Николић и Д. Драшковић: Коришћењем Јава сервлета написати *html* страну која преко линка позива први сервлет, затим прослеђује на други сервлет методом *sendRedirect*, а други сервлет правити форму за унос имена корисника и дугме за потврду, па се позива трећи сервлет. Трећи сервлет чита податак (*GET* параметар) и прослеђује том сервлету од стране другог

4.3.1 Корак 1. избор назива фолдера и фајлова

Направићемо фолдер са називом *servlet02* у *webapps* директоријуму на серверу *tomcat* у коме је сервлет и сви његови фолдери и фајлови, као у Табели 4.3.1.

Табела 4.3.1 Структура директоријума сервера (d:\myProject\ tomcat)



Поновићемо најважније детаље:

- Произвољно бирамо име фолдера у коме су сви сервлет фолдери и фајлови, "servlet02".
- Радни директоријум сервера је одређен при инсталирању сервера, у овом примеру "D:\myProject\tomcat"
- Сервер очекује да фолдери са сервлетима буду испод радног директоријума, у фолдеру "webapps", где су и *html* фајлови, "D:\myProject\tomcat\webapps".
- Сервлет је у овом примеру у фолдеру "servlet02", и путања је "D:\myProject\tomcat\webapps\servlet02".

4.3.2 Корак 2. Направити *index.html* фајл који ће позвати сервлет

У фолдеру "D:\myProject\tomcat\webapps\servlet02" треба да се направи "index.html".

Путања до фајла је следећа "D:\myProject\tomcat\webapps\servlet02\index.html".

Основна минимална структура фајла је

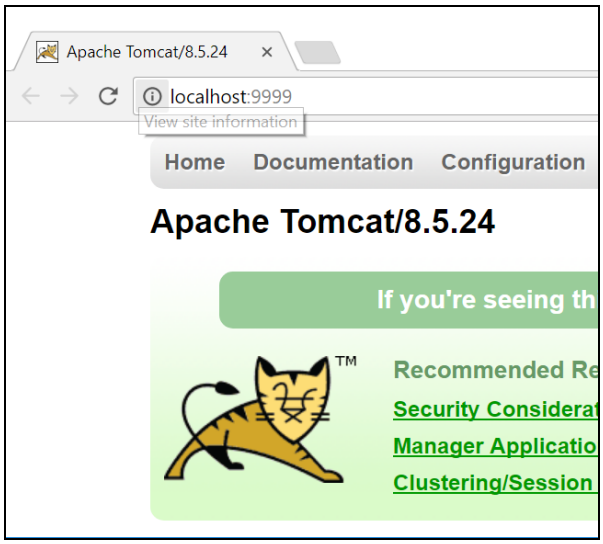
```
<html>
  <head> <title> … </title> </head>
  <body> potrebno znanje html </body>
</html>
```

Ради комплетности, за овај пример следи направљени *html* фајл:

```
<!DOCTYPE html>
<html>
  <head>
    <title>prosledjivanje servleta</title>
  </head>
```

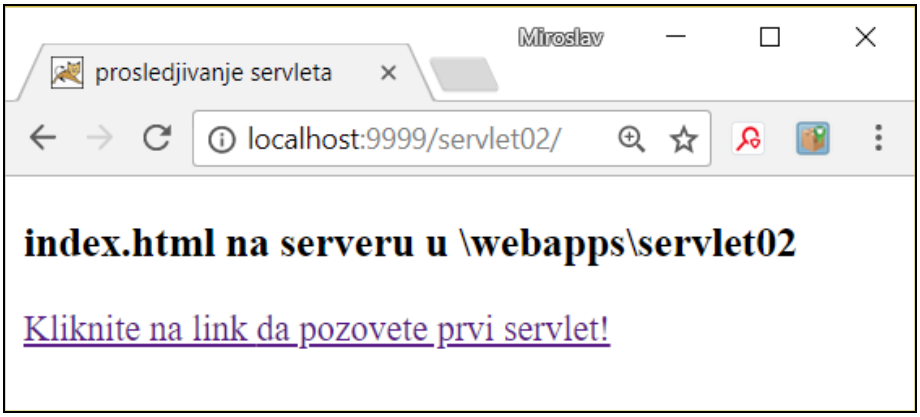
```
<body>
  <h3>index.html na serveru u \webapps\servlet02</h3>
  <p><a href="prviServlet">Kliknite na link
          da pozovete prvi servlet!</a></p>
  <!-- pozivom servleta preko linka
       poziva se doGet() metod-->
</body>
</html>
```

Паралелно са прављењем сервлета радимо тестирање. Стартује се сервер, покрене прегледач. У прегледач се унесе адреса "http://localhost:9999/". Ако је све у реду, добија се поздравни екран сервера, као на слици 4.3.1.



Слика 4.3.1 Провера да је сервер стартован

Унесе се адреса где треба да буде сервлет "http://localhost:9999/servlet02/". У овом серверовом фолдеру се налази фајл "index.html", и прегледач интерпретира садржај дајући приказ на клијентовој страни као на слици 4.3.2.



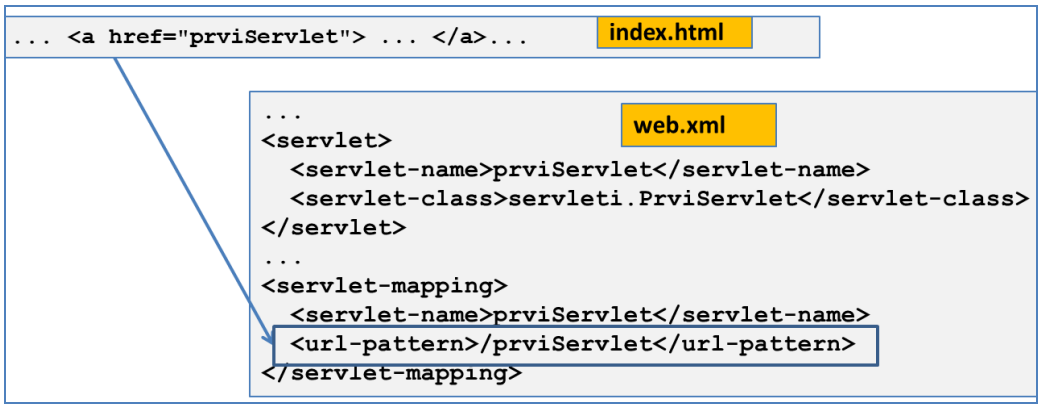
Слика 4.3.2 Приказ *index.html*

Из садржаја "index.html", препознајемо да се сервлет позива преко линка користећи *doGet()* методу.

Након клика на линк, методом "doGet()" додаје се адреси текст "prviServlet", тако да у адреси која се шаље серверу пише "http://localhost:9999/servlet02/prviServlet".

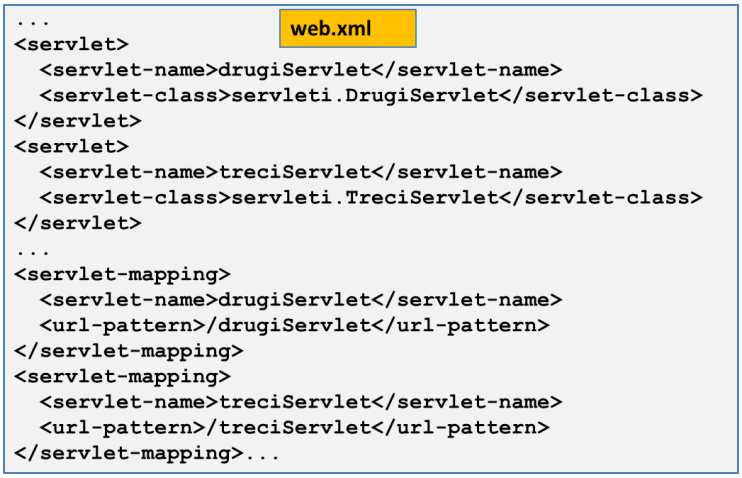
4.3.3 Корак 3. Направити *web.xml*

На основу онога што пише у *index.html* фајлу знамо само где се налази први сервлет и ништа више од тога. Сервер у *web.xml* треба да пронађе назив "prviServlet" на основу кога треба да одреди мапирање у Јава класу.



Слика 4.3.3 Сервер у *web.xml* треба да пронађе назив из *index.html* који користи за мапирање

У овом задатку треба да се направе три сервлета, и за сваки од њих треба да се одреде називи који ће обезбедити коректно мапирање у различите јава класе. Осим мапирања, у овом *web.xml* фајлу не зна се ни ко ће, ни када, позвати остале сервлете.



Слика 4.3.4 Сервер у *web.xml* треба да пронађе мапирања за три сервлета

Ради комплетности, у наставку је комплетан код за *web.xml*, који се налази у фолдеру "D:\myProject\tomcat\webapps\servlet02\WEB-INF\web.xml":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="3.0"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

<servlet>
  <servlet-name>prviServlet</servlet-name>
  <servlet-class>servleti.PrviServlet</servlet-class>
</servlet>
<servlet>
  <servlet-name>drugiServlet</servlet-name>
  <servlet-class>servleti.DrugiServlet</servlet-class>
</servlet>
<servlet>
  <servlet-name>treciServlet</servlet-name>
  <servlet-class>servleti.TreciServlet</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>prviServlet</servlet-name>
  <url-pattern>/prviServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>drugiServlet</servlet-name>
  <url-pattern>/drugiServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>treciServlet</servlet-name>
```

```
  <url-pattern>/treciServlet</url-pattern>
</servlet-mapping>
<session-config>
<session-timeout> 1 </session-timeout>
</session-config>
<welcome-file-list>
<welcome-file>index.html</welcome-file>
</welcome-file-list>
</web-app>
```

Важно је запазити да се класе налазе у новом поддиректоријуму `servleti.` Уместо једне класе, у поддиректоријуму треба да буду смештене три Јава класе.

4.3.4 Корак 4. Направити *.java* фајлове

За три Јава класе потребна су три јава фајла. Уместо једног Јава фајла који је био у фолдеру *classes*, сада ће бити поддиректоријум *servleti*, у фолдеру *classes*, а у фолдеру *servleti* ће бити изворни Јава кодови и њихове компајлиране верзије. Овај фолдер се назива пакет, *package servleti*.

Иако се у овим примерима Јава програми и компајлиране класе налазе у истом фолдеру, то не мора да буде касније и на серверу. Програмери често дају иста или слична имена фајловима, па када треба да се уради нека измена у Јава програму, добро је да програмер зна да ли су то изворне верзије које су у истом фолдеру са компајлираним класама. У пракси, изворне и компајлиране верзије треба да буду у различитим фолдерима, зато што сервлети користе искључиво компајлиране верзије а не и изворни Јава код.

Без објашњења садржаја Јава програма, следе све три верзије.

PrviServlet.java има функцију да генерише одговарајући *html* фајл за прегледач. Треба да се уради редирекција на следећи сервлет (*drugiServlet*):

```
package servleti;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class PrviServlet extends HttpServlet {
protected void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
PrintWriter izlaz = response.getWriter();
try {
izlaz.println("<html>");
izlaz.println("<head>");
izlaz.println("<title>Servlet prviServlet</title>");
izlaz.println("</head>");
izlaz.println("<body bgcolor=\"ccff66\">");
izlaz.println("<h1>Ovo je rezultat prvog!</h1>");
izlaz.println("</body>");
izlaz.println("</html>");
} finally {
}
```

```
response.sendRedirect("drugiServlet");
}
}
```

DrugiServlet.java генерише одговарајући *html* фајл за прегледач. Прави се форма за унос података и упућује на трећи сервлет,
`<form action=\"treciServlet\"method=\"get\">.`

```
...
public class DrugiServlet extends HttpServlet {
protected void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html;charset=UTF-8");
PrintWriter izlaz = response.getWriter();
try {
izlaz.println("<html>");
izlaz.println("<head>");
izlaz.println("<title>Servlet drugiServlet</title>");
izlaz.println("</head>");
izlaz.println("<body bgcolor=\"ffcc00\">");
izlaz.println("<h1>Ovo je rezultat drugog servleta, a ne prvog!</h1>");
izlaz.println("<form action=\"treciServlet\"method=\"get\">");
izlaz.println("Unesite ime: <input type=\"text\"name=\"ime\">");
izlaz.println("<input type=\"submit\" value=\"Pozovi treci servlet\">");
izlaz.println("</form>");
izlaz.println("</body>");
izlaz.println("</html>");
} finally {
izlaz.close();
}}}
```

TreciServlet.java преузима податке и генерише *html* фајл за прегледач.

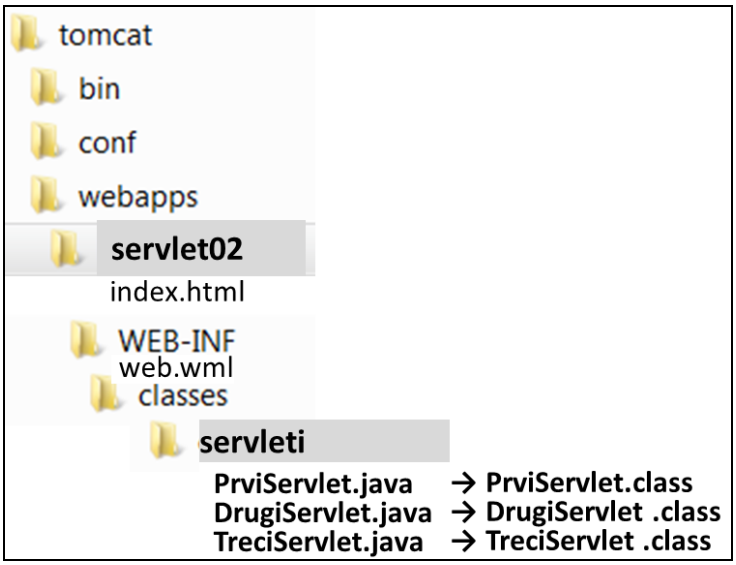
Приказује се порука шта је урадио, и има линк на *html* форму која је у *index.html*.

```
...
public class TreciServlet extends HttpServlet {
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
try {
String ime=request.getParameter("ime");
out.println("<html>");
out.println("<head>");
out.println("<title>Servlet treciServlet</title>");
out.println("</head>");
out.println("<body bgcolor=\"99ccff\">");
out.println("<h1>Dobrodosli " + ime+ "</h1>");
out.println("<br/>");
out.println("<a href=\"index.html\">Povratak na prvu stranu!</a>");
out.println("</body>");
```

```
out.println("</html>");
} finally {
out.close();
}}}
```

4.3.5 Корак 5. Компајлирањем добити *.class* фајл

Из програма у јава формату генеришу се *class* фајлови компајлирањем јава фајлова. Компајлирање се може радити припремом *.bat* фајлова, или коришћењем развојних окружења као што су Еклипс, с тим да се сви *.class* фајлови поставе у фолдер *classes*. У овом примеру, сви *.java* и *.class* фајлови су у фолдеру *servleti*.



Слика 4.3.5 Графички приказ фолдера и фајлова

Слика 4.3.5 илуструје где се који фајл и фолдер налази у овом примери. Стрелица показује да се који се *.class* фајл добија од *.java* фајла.

4.3.6 Корак 6. Стартовати сервер

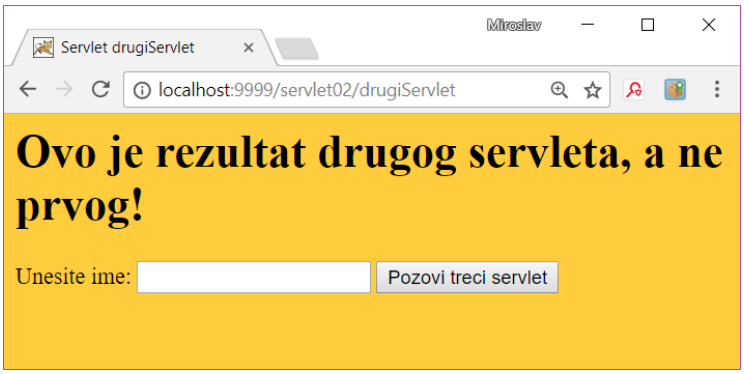
За стартовање сервер може да се користи command prompt, програма сервер менаџер, или развојно окружење Еклипс.

4.3.7 Корак 7. Тестирање рада апликације

За тестирање апликације, покрене се сервер и у прегледачу се унесе адреса, http://localhost:9999, када се добија поздравни екран са сервера, што значи да је сервер покренут и да исправно ради. Уношењем адресе у прегледа на клијентовој страни "http://localhost:9999/servlet02/", сервер приказује садржај фајла "index.html", као што је већ приказано на слици 4.3.2 (подразумевани фајл је *index.html*).

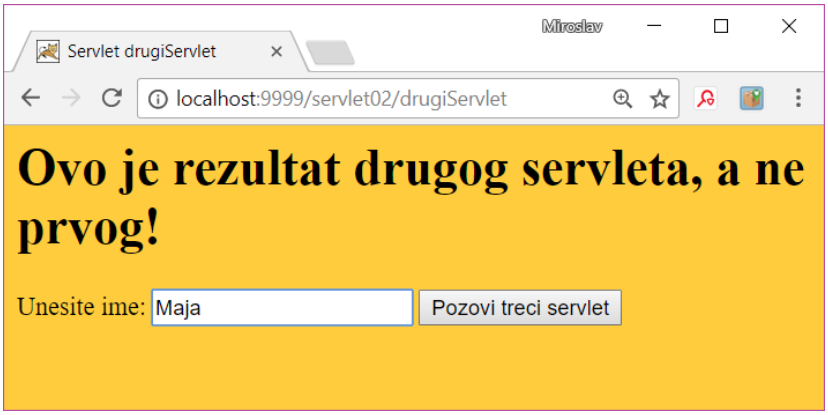
После клика на линк *html* фајла, позива се први сервлет, али он одмах позива други сервлет. Практично, први сервлет служи да уради редирекцију на други сервлет и не даје никакав приказ у прегледачу.

Прегледач приказује садржај *html* фајла, као што је илустровано на слици 4.3.6. При чему је адреса у прегледачу "http://localhost:9999/servlet02/drugiServlet".



Слика 4.3.6 Прегледач приказује садржај *html* фајла који генерише други сервлет

Други сервлет генерише *html* фајл за клијентов прегледач који приказује поруку и даје форму за унос имена; након уноса имена, кликом на дугме, отвара се трећи сервлет. Мапирања одређује *web.xml*, а шта се приказује у прегледачу, одређују јава класе.



Слика 4.3.7 Попуњена форма након чега се подаци шаљу серверу кликом на дугме

Садржај *html* фајла који је добијен за слику 4.3.5 је следећи

```
<html>
<head><title>Servlet drugiServlet</title></head>
<body bgcolor="ffcc00">
<h1>Ovo je rezultat drugog servleta, a ne prvog!</h1>
<form action="treciServlet"method="get">
```

```
Unesite ime: <input type="text"name="ime">
<input type="submit" value="Pozovi treci servlet">
</form>
</body>
</html>
```

Трећи сервлет генерише *html* фајл за клијентов прегледач, а у адреси је вредност унета у форму:

"http://localhost:9999/servlet02/treciServlet?ime=Maja"

Трећи сервлет генерише нови *html* фајл

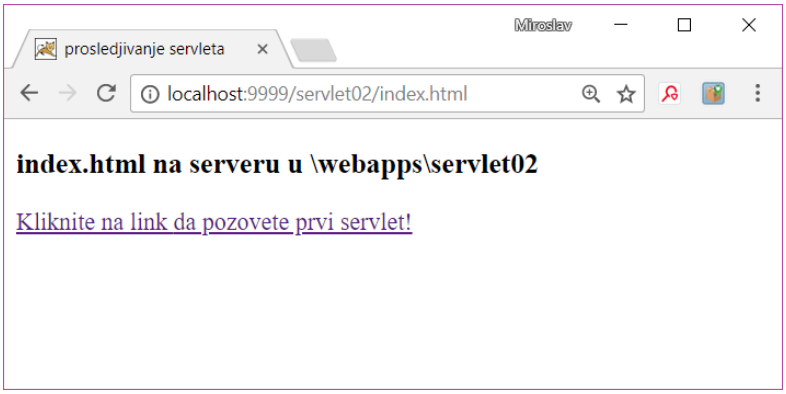
```
<html>
<head>
<title>Servlet treciServlet</title>
</head>
<body bgcolor="99ccff">
<h1>Dobrodosli Maja</h1>
<br/>
<a href="index.html">Povratak na prvu stranu!</a>
</body>
</html>
```

Прегледач интерпретира и приказује као што је на слици 4.3.8.

Слика 4.3.8 Прегледач приказује садржај *html* фајла који генерише трећи сервлет

Кликом на линк, поново се отвара иницијални *index.html*.

Из трећег сервлета се генерише нова адреса која садржи и назив *html* фајла, *index.html*. Све се понавља, први сервлет, па други сервлет, па поново *index.html*. На слици 4.3.9 је поново приказ почетног прозора ове апликације.



Слика 4.3.9 Прегледач поново приказује садржај *index.html*

Ради комплетности, следи *html* код

```
<!DOCTYPE html>
<html>
  <head>
    <title>prosledjivanje servleta</title>
  </head>
<body>
  <h3>index.html na serveru u \webapps\servlet02</h3>
  <p><a href="prviServlet">Kliknite na link
          da pozovete prvi servlet!</a></p>
  <!-- pozivom servleta preko linka
       poziva se doGet() metod-->
</body>
</html>
```

Треба се подсетити да је адреса у прегледачу

"http://localhost:9999/servlet02/index.html"

а на серверу

"D:\myProject\tomcat\webapps\servlet02\index.html"

Да би се видело шта је направио неки од сервлета, и приказали изворни кодови који користи прегледач може се користити:

Open menu → Web Developer → Page Source

Први сервлет нема приказ у прегледачу, већ само уради редирекцију на други сервлет.

drugiServlet

```
<html>
<head><title>Servlet drugiServlet</title></head>
<body bgcolor="ffcc00">
<h1>Ovo je rezultat drugog servleta, a ne prvog!</h1>
<form action="treciServlet"method="get">
Unesite ime: <input type="text"name="ime">
<input type="submit" value="Pozovi treci servlet">
</form>
</body>
</html>
```

treciServlet

```
<html>
<head>
<title>Servlet treciServlet</title>
</head>
<body bgcolor="99ccff">
<h1>Dobrodosli Maja</h1>
<br/>
<a href="index.html">Povratak na prvu stranu!</a>
</body>
</html>
```

Index.html

```
 <!DOCTYPE html>
<html>
  <head>
    <title>prosledjivanje servleta</title>
  </head>
<body>
  <h3>index.html na serveru u \webapps\servlet02</h3>
  <p><a href="prviServlet">Kliknite na link
          da pozovete prvi servlet!</a></p>
  <!-- pozivom servleta preko linka
       poziva se doGet() metod-->
</body>
</html>
```

4.3.8 Комплетан поступак рада сервлета

Поновићемо цео поступак рада сервлета у овом примеру.

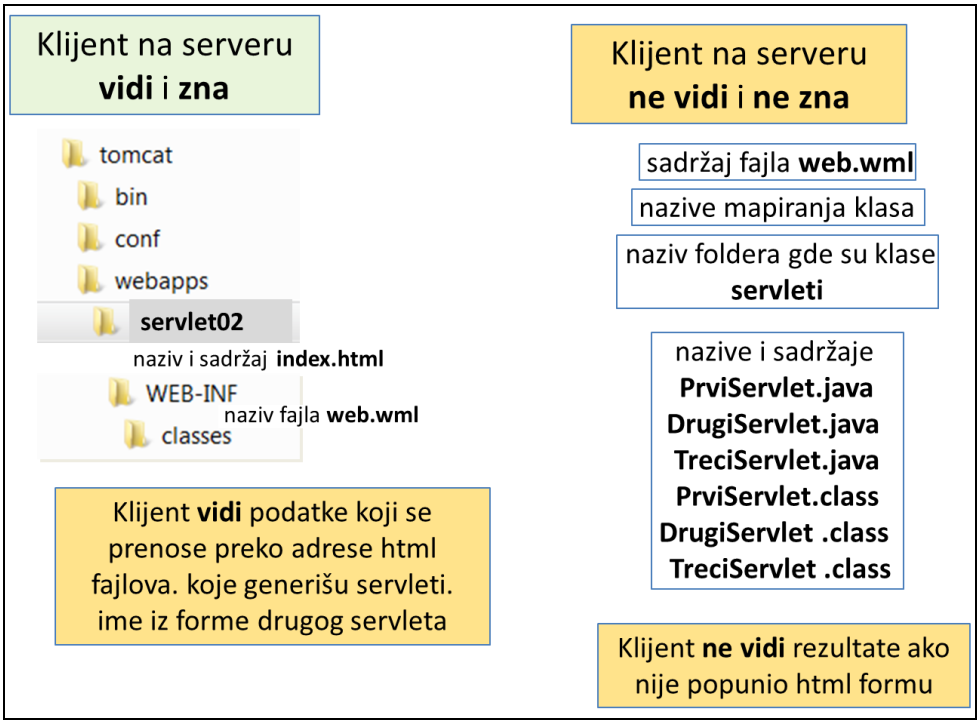
1. Клијент уноси адресу на серверу коју жели да посети, http://localhost:9999/servlet02/.
2. Ако у адресу није унет *html* фајл, подразумевани фајл је *index.html*, адреса као да је "http://localhost:9999/servlet02/index.html"; а на серверу се налази у "D:\myProject\tomcat\webapps\servlet02\index.html".
3. Сервер шаље *html* фајл клијентовом прегледачу, који може да садржи форму за унос података, линк на другу адресу.

4. Клијент кликом на линк, мења адресу прегледача која на серверу позива сервлет, "http://localhost:9999/servlet02/prviServlet", али прегледач ништа не приказује.
5. Први сервлет одмах ради редирекцију на други сервлет, мења адресу прегледача која на серверу позива сервлет, "http://localhost:9999/servlet02/drugiServlet".
6. Други сервлет шаље форму клијентовом прегледачу, и када се унесу подаци (xxx) и кликне на дугме, позива се трећи сервлет, мења се адреса прегледача која на серверу позива сервлет, "http://localhost:9999/servlet02/trećiServlet?ime=xxx".
7. Мапирање сервлета у одговарајуће класе је дефинисано са "web.xml". Сервер у фајлу "D:\myProject\tomcat\webapps\servlet02\WEB-INF\web.xml" преко *URL* адресе проналази назив сервлета и њему мапирану класу: за "/prviServlet" је "/servleti/PrviServlet.class", за "/drugiServlet" је "/servleti/DrugiServlet.class", за "/treciServlet" је "/servleti/TreciServlet.class".
8. Јава класе генерише *html* фајлове који се шаљу клијентовом прегледачу.
9. Прегледач на страни клијента приказује добијени *html* код

4.3.8.1 Видљивост фолдера и фајлова на серверу

1. Клијент види фолдер који садржи сервлет, *servlet02*.
2. Клијент види *html* фајл који позива сервлете, *index.html*.
3. Клијент види у адреси и *html* фајлу реч из адресе која одговара сервлету, "/prviServlet".
4. Клијент зна структуру фолдера на серверу, *tomcat\webapps, webapps\servlet02, servlet02\WEB-INF, WEB-INF\classes*, као и назив конфигурационог фајла *web.xml*.
5. Клијент не види имена која се користе за мапирање адреса у одговарајуће класе (свака класа има своје одговарајуће име).
6. Клијент не види називе класе које се извршавају, *.java* и *.class*, *PrviServlet.class, DrugiServlet.class, TreciServlet.class*.
7. Клијент не види назив фолдера где су класе, /*servleti*.
8. Клијент не види шта раде класе које се извршавају.
9. Клијент не види активности како су програмиране јава класе.
10. Клијент види и приказује *html* садржај добијен од сервера које генеришу класе.
11. Клијент види и приказује податке у адреси *html* фајлова, ако их тако генеришу класе.

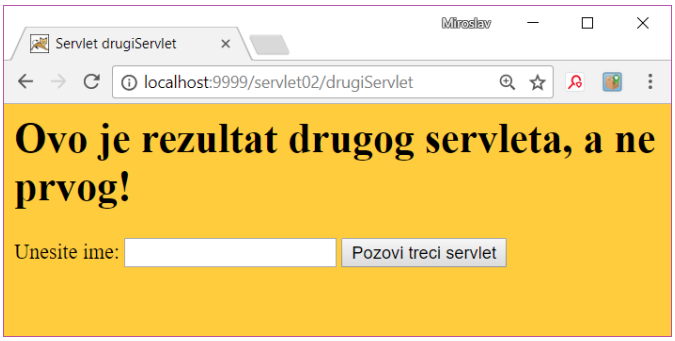
На слици 4.3.10 је графички приказано шта клијент види или зна и шта не види на серверу.



Слика 4.3.10 Графички приказ шта клијент види или зна и шта не види

4.3.9 Ефекти директног уношења назива сервлета

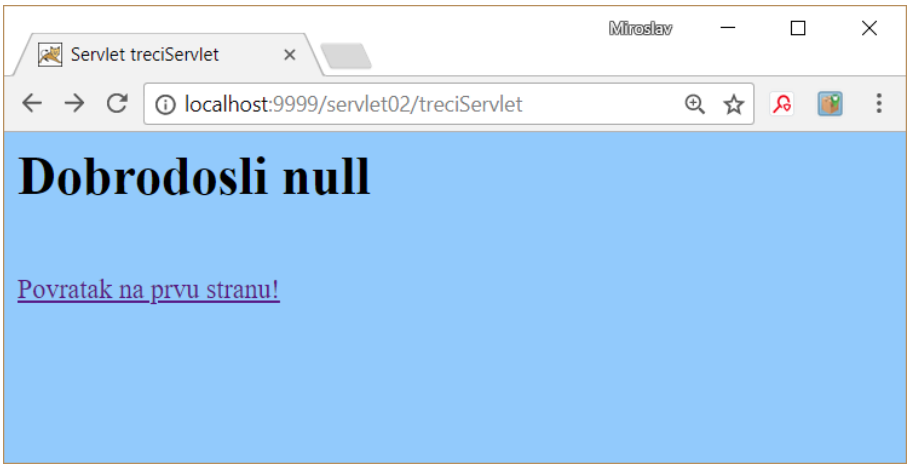
Ако клијент укуца назив другог сервлета, "http://localhost:9999/servlet02/drugiServlet" добија се приказ као да је регуларно дошао после првог сервлета, као на слици 4.3.11. Клијент регуларно наставља да се креће на следећи сервлет, или иницијални *index.html*, као да се све регуларно радило.



Слика 4.3.11 Приказ прегледача након директног уноса назива другог сервлета

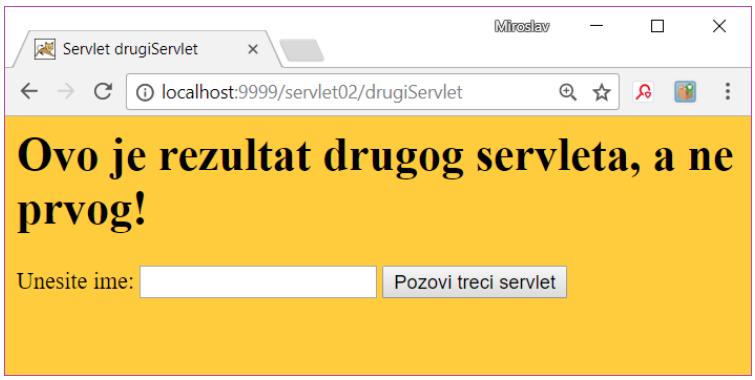
Ако је клијент укуцао назив трећег сервлета, "http://localhost:9999/servlet02/treciServlet", добија се приказ као да је регуларно дошао са другог без уноса података у форму, као што се види на слици 4.3.12, где се добија за име *null*.

Клијент регуларно наставља да се креће на следећи сервлет, или иницијални *index.html*, као да је све регуларно радило.



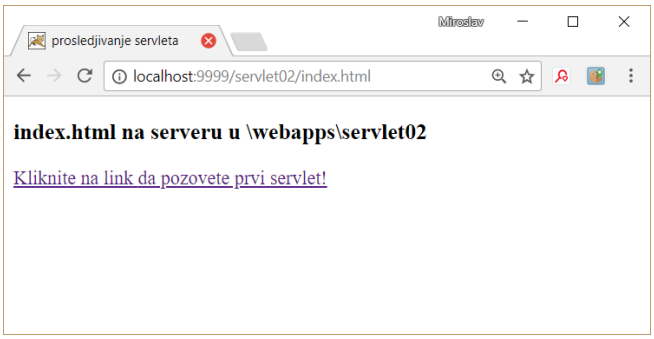
Слика 4.3.12 Приказ прегледача након директног уноса назива трећег сервлета

Ако клијент укуца назив првог сервлета, "http://localhost:9999/servlet02/prviServlet", добија приказ другог сервлета, и адресу другог сервлета у прегледачу, као да је регуларно дошао после редирекције са првог сервлета, као на слици 4.3.13. Клијент регуларно наставља да се креће на следећи сервлет, или иницијални *index.html*, као да је све регуларно рађено.



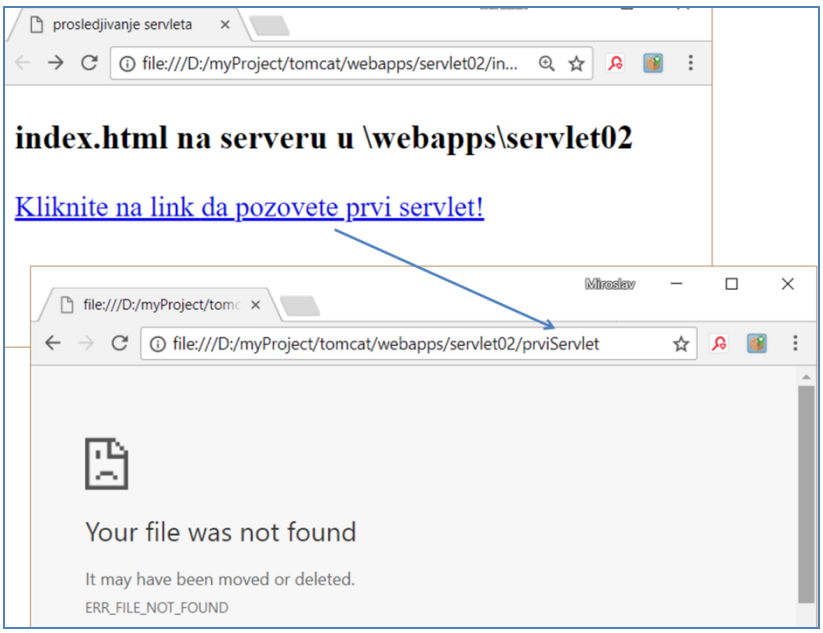
Слика 4.3.13 Приказ прегледача након директног уноса назива првог сервлета

Било где да је започет рад позивањем сервлета, долази се поново до иницијалног *index.html*, "http://localhost:9999/servlet02/index.html", као на слици 4.3.14.



Слика 4.3.14 Прегледач приказује садржај *index.html* без обзира од ког сервлета се кренуло

Ако се кликне на *index.html* фајл и отвори на директно на серверу (у адреси се види *file ...*), без позивања фолдера где су сервлети, тада ће бити приказан у прегледачу *index.html*, али неће радити сервлети, зато што на овај начин прегледач не види *web.xml*, па не зна мапирања позива у класе. .



Слика 4.3.15 Прегледач приказује садржај *index.html* али не раде сервлети без мапирања

За исправан рад потребно је да постоји комуникација између клијента и сервера, овај приступ да су и клијент и сервер на истом рачунару треба да омогући бржи развој веб апликација.

4.3.10 Питања и задаци за вежбу

1. Направити неколико сервлета који ће поновити пример из ове теме, с тим да се у *html* форми уносе два броја а један сервлет треба да израчуна збир, и врати резултат сабирања.
2. Написати извештај са описом евентуалних проблема.

5. Литература

A.1. [2008_Nikolic_IP] Бошко Николић, **Интернет програмирање 1**, ВИШЕР, Београд, (2008).

A.2. [2015_Cirovic_UuIT] Зоран Ћировић, **Увод у Интернет технологије**, ВИШЕР, Београд, (2015).

A.3. [2017_Nikolic_PIAuszz] Бошко Николић, Дражен Драшковић, **Програмирање интернет апликација, уџбеник са збирком задатака**, Академска мисао, Београд, (2017).

A.4. [2018_Djenić_OP], Слободанка Ђенић, **Основи програмирања**, ВИШЕР, Београд, (2018).

A.5. [2018_Strbac_pUOP] Светлана Штрбац-Савић, **Приручник за лабораторијске вежбе Увод у објектно програмирање**, ВИШЕР, (2018)

A.6. [2019_Strbac_OP2] Перица Штрбац, **Објектно Програмирање 2**, ВИШЕР, Београд, (2019).

A.7. [2019_Strbac_pVD] Светлана Штрбац-Савић, Нада Сталетић, Бранислав Меанџија, **Веб дизајн - Приручник за лабораторијске вежбе**, ВИШЕР, Београд, 2019.

A.8. [2020_apache.org_ASF] https://www.apache.org, **Apache Software Foundation**, (2020).

A.9. [2020_apache.org_DAHS] https://httpd.apache.org/download.cgi, **Downloading the Apache HTTP Server**, (2020).

A.10. [2020_ntu.edu.sg_HiATgsJsp] https://www.ntu.edu.sg/home/ehchua/programming/howto, **How to install Apache Tomcat and get started with Java servlet programming**, (2020).

A.11. [2020_tomcat.apache.org_AT] http://tomcat.apache.org, **Apache Tomcat**, (2020).

A.12. [2020_tutorialspoint.com/jsp_JSPT] https://www.tutorialspoint.com/jsp/index.htm, **JSP Tutorial**, (2020).

A.13. [2020_w3schools.com_CSST] https://www.w3schools.com/css, **CSS Tutorial**, (2020).

A.14. [2020_w3schools.com_CSSR] https://www.w3schools.com/cssref, **CSS Reference**, (2020).

A.15. [2020_w3schools.com_HTMLER] https://www.w3schools.com/tags, **HTML Element Reference**, (2020).

A.16. [2020_w3schools.com_HTMLT] https://www.w3schools.com/html, **HTML Tutorial**, (2020).

A.17. [2020_w3schools.com_JavaScript and JSHDR] https://www.w3schools.com/jsref, **JavaScript and HTML DOM Reference**, (2020).

A.18. [2020_w3schools.com_JST] https://www.w3schools.com/js, **JavaScript Tutorial**, (2020).

A.19. [2020_w3schools.com_WDR] https://www.w3schools.com/whatis, **Web Development Roadmaps**, (2020).

Кључне речи

веб апликације, интернет апликације, Јава веб технологија, Јава програмски језик, клијент-сервер архитектура, објектно-оријентисани програмски језик, *CSS* шаблони, *HTML*, *JavaScript*, *Server-side* скриптинг, *XML*.

Индекс појмова

Поговор

Уџбеник “Интернет програмирање” намењен је студентима Високе школе електротехнике и рачунарства струковних студија у Београд који овај предмет слушају као изборни на студијским програмима Нове рачунарске технологије, Информациони системи, Електроника и телекомуникације и Аутоматика и системи управљања возилима, а који су акредитовани 2017. године. Циљ предмета је оспособљавање студената да пројектују и пишу савремене Интернет апликације коришћењем Јава програмског језика.

“Интернет програмирање” је био изборни предмет и у наставним плановима студијских програма који су били акредитовани 2012. године. Циљ наставе је био оспособљавање студената да пројектују и пишу основне веб апликације, коришћењем елемената *HTML, CSS, JavaScript* и коришћењем Јава веб технологија (*JSP/Servlet*). Исход предмета је био да након положеног предмета студенти буду оспособљени да уз помоћ савременог развојног окружења развију комерцијалне Интернет апликације користећи програмски језик Јава. Теоријска настава је имала неколико целина: Увод у веб програмирање, *HTML, CSS, JavaScript*, основи објектно-оријентисаних језика и програмског језика Јава, *JSP/Servlet* технологије.

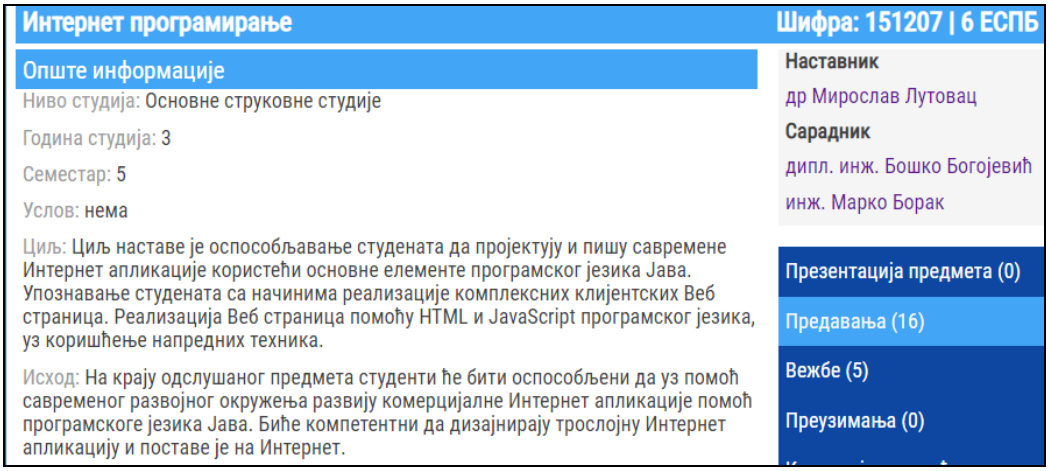
“Интернет програмирање” спада у групу предмета која се под различитим називима може срести у студијским програмима Софтверско инжењерство. Софтверско инжењерство је област која пружа студентима могућност брзог проналажења посла и добре зараде, па тако привлачи све већи број заинтересованих студената, и све веће могућности за реализацију пројеката у скоро свим сегментима живота. Нове верзије софтвера и алата за развој апликација постављају нове захтеве и за програмере који раде у области интернет апликација. Тако је иновиран и садржај “Интернет програмирање” који од акредитације 2017. године има атрактивније целине које се изучавају у теоријској настави: (1) Основни појмови и термини; (2) Архитектура вишеслојних клијент-сервер система; (3) Карактеристике и структура интернет апликација; (4) Интернет протоколи; (5) Језици за опис стране, *XML, HTML, DOM, JavaScript, Server-side* скриптинг; (6) Развојне платформе за интернет апликације; (7) Јава апликација, аплети; *JSP, EJB, JSP*; (8) Рад са базама података – *JDBC* (*Java DataBase Connectivity*); (9) Употреба *STRUTS framework*-а; (10) Рад са веб формуларима и корисничка интеракција; (11) *JCF* (*Java collections framework*); (12) Веб сервиси; (13) Проблем сигурности (сесије, аутентификација и ауторизација, енкрипција, инфраструктура јавног кључа, *HTTPS*). Практична настава прати програм предавања и одвија се демонстрирањем комерцијалних апликација применом наведених технологија. По новом програму “Интернет програмирање” се предаје од школске 2019/2020. године.

Слични курсеви постоје на другим факултетима на више студијских програма.

На Електротехничком факултету Универзитета у Београду, предмет Интернет програмирање се држао у школској 2018/2019. години као изборни предмет на Одсеку за сигнале и системе и Одсеку за електронику, на четвртој години основних академских студија, а настава се изводила на рачунарима у рачунарској лабораторији. Циљ наставе је био да се студенти оспособе да пројектују и пишу савремене интернет апликације, да се упознају са начинима реализације комплексних клијентских веб страница помоћу језика *HTML, CSS* шаблона и *JavaScript* програмског језика и уз коришћење напредних техника, као и са реализацијом трослојних веб апликација коришћењем програмског језика *PHP*, *MySQL* базе података и *AJAX* технологије. Градиво првог дела курса је садржало увод у *HTML, HTML* листе, линкови и табеле, *HTML* форме, фрејмови и *layout, CSS* (*Cascading Style Sheet* - рад са шаблонима), и програмски језик *JavaScript*. Студентима су били на располагању задаци (са урађеним програмским кодовима) и збирке задатака из *HTML, CSS*, и *JavaScript*-а. Градиво другог дела курса је покривало програмски језик *PHP*, рад са *MySQL* базом података, *AJAX* технологију, са задацима (са урађеним програмским кодовима) и припремљеном базом. Израдом пројекта студенти су демонстрирали стечено знање из свих области које су по плану овог предмета.

На Електротехничком факултету Универзитета у Београду, предмет Програмирање интернет апликација се држао у школској 2018/2019. години као обавезни предмет на Одсеку за софтверско инжењерство на четвртој години основних академских студија, као и на Одсеку за рачунарску технику и информатику на трећој години основних академских студија. Циљ курса је био да се студенти упознају са основним појмовима развоја вишеслојних Интернет апликација базираних на програмском језику *Java*. Студенти су се најпре упознавали са најсавременијим технологијама за развој и имплементацију интернет апликација, а затим су се оспособљавали да пројектују и развију вишеслојне сложеније интернет апликације користећи модерне начине и савремене веб технологије. На крају курса студенти показују да су освојили знање израдом пројекта, и да су тако спремни за развој интернет апликација какве се захтевају на тржишту. Садржај курса је зависио од изабраног студијског програма а обухватао је неке од тема*: Java Servlet* технологија, *Java Server Faces* (*JSF*), *Facelets, AJAX* (*Asynchronous JavaScript and XML*), *Angular, Frameworks* (*Spring, Hibernate ORM, Primefaces*), *HTML/HTML*5, *CSS* (*Cascading Style Sheets*), *JavaScript*, повезивање са базама података из Јаве, *Java Server Pages* (*JSP*), *Java Server Faces* (*JSF*). Студентима су на располагању били урађени задаци (са урађеним програмским кодовима) и збирком задатака за вежбе.

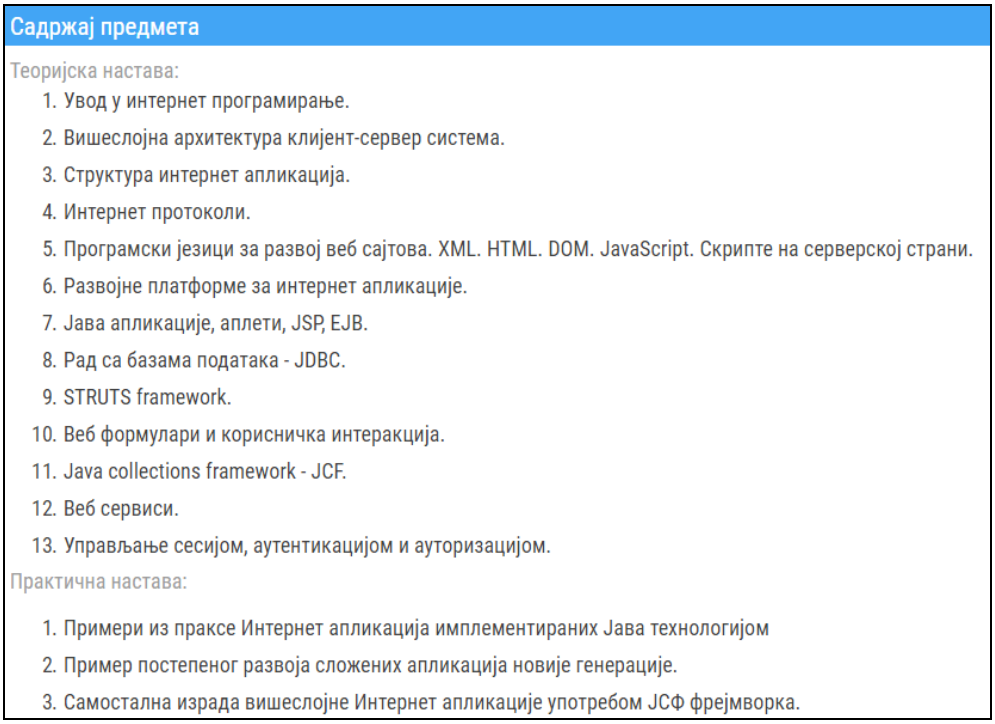
Опште информације о предмету “Интернет програмирање” на сајту Високе школе електротехнике и рачунарства струковних студија у Београд на студијском програму Нове рачунарске технологије, https://www.viser.edu.rs, јануар 2019., приказане су на Слици 1.



Слика 1. Опште информације о предмету на сајту https://www.viser.edu.rs, за 2019/2020.

Садржај предмета "Интернет програмирање" који је акредитован за 2019/2020 приказан је на Слици 2. Опште информације о предмету "Интернет програмирање" на сајту Високе школе електротехнике и рачунарства струковних студија у Београд на студијском програму Нове рачунарске технологије по плану и програму 2018/2019, приказан је на Слици 3. Садржај предмета "Интернет програмирање" који је реализован по плану и програму за 2018/2019 приказан је на Слици 4.

На основу планова и програма који су били акредитовани или су акредитовани од 2017. године, а који се односе на "Интернет програмирање", у овом уџбенику су обрађени материјали који су коришћени у настави на Високој школи електротехнике и рачунарства струковних студија у Београд тако да покривају градиво по обе акредитације из предмета "Интернет програмирање". Делови градива који су специфични и нису у функцији развоја програма, а обрађују се у другим предметима, нису детаљно обрађени како уџбеник не би био превише опширан. Уџбеник садржи детаљно урађене примере са циљем да студент има сво потребно знање пре него што почне да решава сложеније задатке.



Садржај предмета

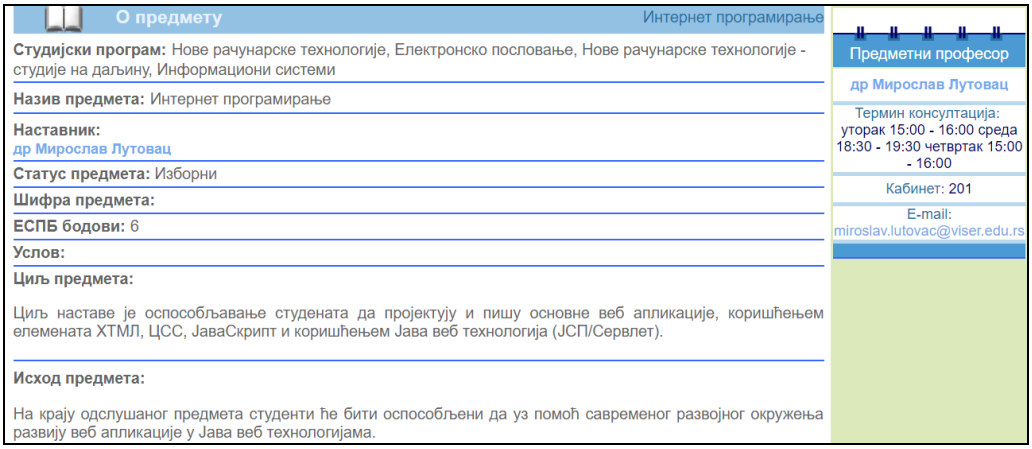
Теоријска настава:

1. Увод у интернет програмирање.
2. Вишеслојна архитектура клијент-сервер система.
3. Структура интернет апликација.
4. Интернет протоколи.
5. Програмски језици за развој веб сајтова. XML. HTML. DOM. JavaScript. Скрипте на серверској страни.
6. Развојне платформе за интернет апликације.
7. Јава апликације, аплети, JSP, EJB.
8. Рад са базама података - JDBC.
9. STRUTS framework.
10. Веб формулари и корисничка интеракција.
11. Java collections framework - JCF.
12. Веб сервиси.
13. Управљање сесијом, аутентикацијом и ауторизацијом.

Практична настава:

1. Примери из праксе Интернет апликација имплементираних Јава технологијом
2. Пример постепеног развоја сложених апликација новије генерације.
3. Самостална израда вишеслојне Интернет апликације употребом ЈСФ фрејмворка.

Слика 2. Садржај предмета “Интернет програмирање” акредитован за 2019/2020.



О предмету — Интернет програмирање

Студијски програм: Нове рачунарске технологије, Електронско пословање, Нове рачунарске технологије - студије на даљину, Информациони системи

Назив предмета: Интернет програмирање

Наставник: др Мирослав Лутовац

Статус предмета: Изборни

Шифра предмета:

ЕСПБ бодови: 6

Услов:

Циљ предмета:

Циљ наставе је оспособљавање студената да пројектују и пишу основне веб апликације, коришћењем елемената ХТМЛ, ЦСС, ЈаваСкрипт и коришћењем Јава веб технологија (ЈСП/Сервлет).

Исход предмета:

На крају одслушаног предмета студенти ће бити оспособљени да уз помоћ савременог развојног окружења развију веб апликације у Јава веб технологијама.

Предметни професор

др Мирослав Лутовац

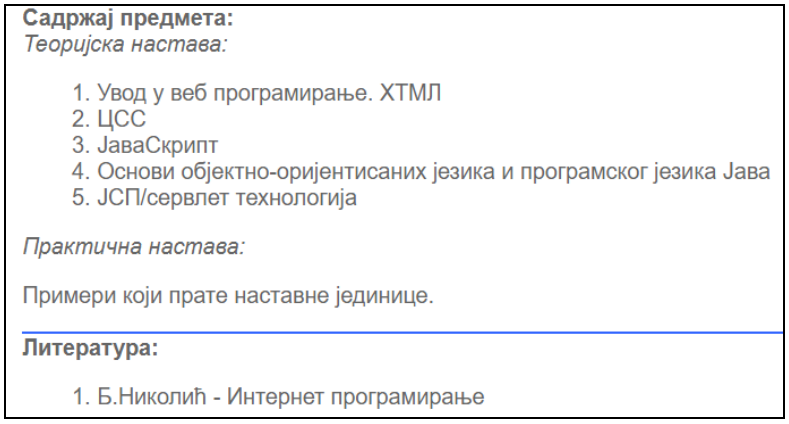
Термин консултација: уторак 15:00 - 16:00 среда 18:30 - 19:30 четвртак 15:00 - 16:00

Кабинет: 201

E-mail: miroslav.lutovac@viser.edu.rs

Слика 3. Опште информације о предмету на сајту https://www.viser.edu.rs, за 2018/2019.

Уџбеник “Интернет програмирање” следи теоријски и садржајно исте или сличне предмете које је држао професор Бошко Николић у ВИШЕР-у и на Електротехничком факултету у Београду, као и уџбенике које је објавио професор Бошко Николић.


Садржај предмета:
Теоријска настава:

1. Увод у веб програмирање. ХТМЛ
2. ЦСС
3. ЈавaСкрипт
4. Основи објектно-оријентисаних језика и програмског језика Јава
5. ЈСП/сервлет технологија

Практична настава:

Примери који прате наставне јединице.

Литература:

1. Б.Николић - Интернет програмирање

Слика 4. Садржај предмета “Интернет програмирање” за 2018/2019.

Уџбеник “Интернет програмирање” је написан након што су одржана предавања првој генерацији по акредитованом студијском програму 2019/2020.

5. јуни 2020. У Београду

Др Мирослав Д. Лутовац, дипл. инж.,
професор Високе школе електротехнике и рачунарства струковних студија у Београду,
научни саветник, дописни члан Академије инжењерских наука Србије.