

Светлана Штрбац Савић
Душан Чоко

Приручник за лабораторијске вежбе
Увод у објектно програмирање

Висока школа електротехнике и рачунарства струковних студија у
Београду, 2018.

Аутори:	Светлана Штрбац-Савић, Душан Чоко
Рецензенти:	Мирослав Лутовац, Слободанка Ђенић
Лектор:	
Корице:	Душан Чоко
Издавач:	Висока школа електротехнике и рачунарства струковних студија у Београду
Штампа:	
Тираж:	200
Издање:	прво

ISBN 978-86-7982-275-8

CIP – Каталогизација у публикацији
Народна библиотека Србије, Београд

ШТРБАЦ-САВИЋ, Светлана 1972 -
Приручник за лабораторијске вежбе Увод у објектно програмирање / Светлана Штрбац-Савић, Душан Чоко – 1. издање – Београд: Висока школа електротехнике и рачунарства струковних студија 2018 () – стр: илустр; 29cm

Тираж 200 – Библиографија: стр.

ISBN 978-86-7982-275-8

1. Чоко, Душан, 1990 – [аутор]
а) Програмирање – вежбе б) Програмски језик Visual Basic - задаци
COBISS.SR-ID

Име и презиме	
Број индекса	

Вежба а	Датум	Лаборатори ја	Прегледао	Напомена
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

Предговор

Овај приручник је намењен студентима прве године Високе школе електротехнике и рачунарства струковних студија у Београду, који су изабрали предмет Увод у објектно програмирање као и свима који желе да науче основе објектног програмирања.

Приручник садржи 12 лабораторијских вежби чији је циљ да упознају читаоца са основним знањима програмирања кроз креирање графичких апликација коришћењем Visual Basic програмског језика. Софтверски алат у ком су урађени примери и задаци који се налазе у приручнику је Visual Studio 2017.

Аутори се захваљују рецензентима др Мирославу Лутовцу и др Слободанки Ђенић на корисним предлозима и сугестијама којим су допринели квалитету овог приручника.

Поред највеће пажње током формирања и писања овог приручника могуће је да су се поткрале грешке. Аутори ће бити захвални свим читаоцима који укажу на њих или дају предлоге и сугестије.

Београд, 2018.

Аутори

Садржај

Упознавање са развојним окружењем – Visual Studio 2017 Community Edition	1
Инсталација развојног окружења.....	1
Креирање првог пројекта.....	3
Прозори развојног окружења.....	5
Покретање апликације.....	7
„Zdravo svete“ апликација.....	8
Основни елементи графичког окружења.....	11
Форма.....	11
Лабела.....	13
Поље за унос текста.....	16
Контејнер за слику.....	16
Догађаји и динамичка измена својстава.....	16
Именовање догађаја и елемената.....	17
Дефинисање основних догађаја.....	19
Рад са променљивама.....	23
Декларација и иницијализација променљивих.....	26
Декларација променљивих.....	27
Иницијализација променљивих.....	28
Конзолна апликација.....	28
Аритметички оператори и конверзије вредности.....	30
Опсег видљивости променљивих – локалне и атрибутске.....	32
Math класа.....	34
Наредбе гранања.....	38
If/else наредба.....	38
Тернарна if наредба.....	41
Логички оператори.....	42
Select Case наредба.....	44
RadioButton и CheckBox елементи.....	46
Наредбе циклуса.....	50
Наредбе Continue, Exit.....	52
Низови.....	58
Статички низови.....	58
Динамички низови.....	60

Пролазак кроз низове коришћењем петљи.....	61
For петља.....	61
For Each петља.....	62
Сортирање низова.....	63
Bubble sort.....	63
Selection sort.....	65
Сортирање низа текстова.....	66
Процедуре и модули.....	70
Сложене структуре података.....	79
Класе и објекти.....	79
Наслеђивање класа.....	84
Динамички графички елементи.....	88
Динамичко креирање елемената.....	88
Везивање догађаја за елементе.....	89
Креирање произвољних догађаја.....	90
Везивање више елемената за један догађај.....	93
Рад са ресурсима и датотекама.....	97
My.Computer.....	97
My.Computer.FileSystem објекат.....	98
Читање и упис садржаја у датотеке секвенцијално.....	103
Дијалог прозори, Менији и MDI апликације.....	107
Дијалог прозори.....	107
ColorDialog.....	107
FontDialog.....	108
FolderBrowserDialog.....	109
Менији.....	110
MenuStrip мени.....	110
MDI апликације.....	113
Креирање MDI форме.....	113
Креирање MDI апликације.....	115
Задаци за самостални рад.....	122
Мултимедијални елементи.....	124
Windows Media Player.....	124
Picture box.....	128
Web browser.....	131

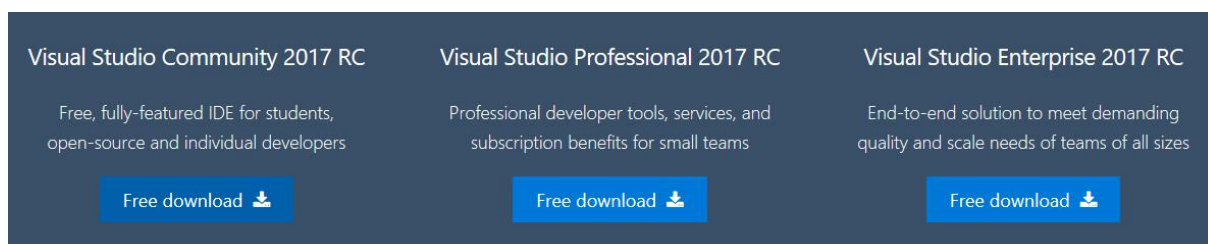
Adobe PDF Reader.....	132
Задаци за самостални рад.....	134
Литература.....	136



Упознавање са развојним окружењем – Visual Studio 2017 Community Edition

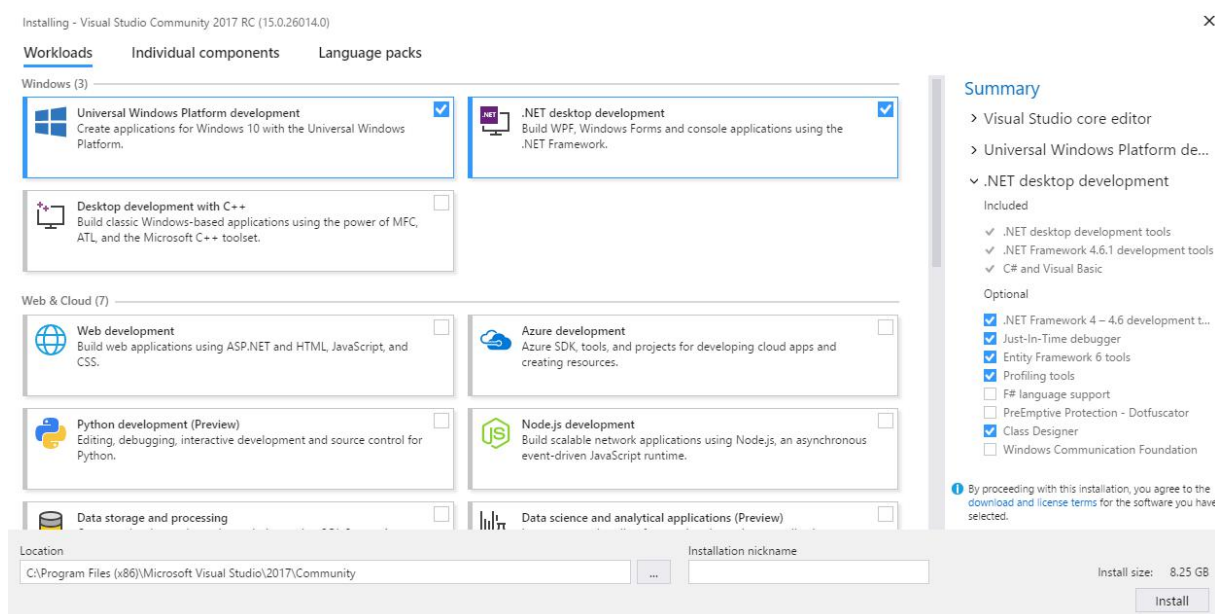
Инсталација развојног окружења

Visual Studio Community Edition је бесплатно развојно окружење које се користи за развој *Visual Basic* апликација. Развојно окружење се може преузети тако што се у било ком претраживачу унесе *Visual Studio 2017 Community Edition* и први резултат би требао да буде *Microsoft* страница за преузимање окружења. На страници треба изабрати жељено окружење и кликнути на дугме *Free Download* (слика 1).



Слика 1. Преузимање инсталације за развојно окружење *Visual Studio*

Када се инсталација покрене, очекује се да корисник дефинише тип инсталације и да изабере опције које је потребно инсталирати (слика 2). Од опција је довољно да изабрати *.NET desktop development*. Уколико желите да користите и програмски језик *C++* изаберите и опцију *Desktop development with C++*. За поље *Location* треба изабрати локацију где ће се инсталирати програм. Затим треба кликнути на дугме *Install*.



Слика 2. Почетни екран инсталације *Visual Studio 2017*

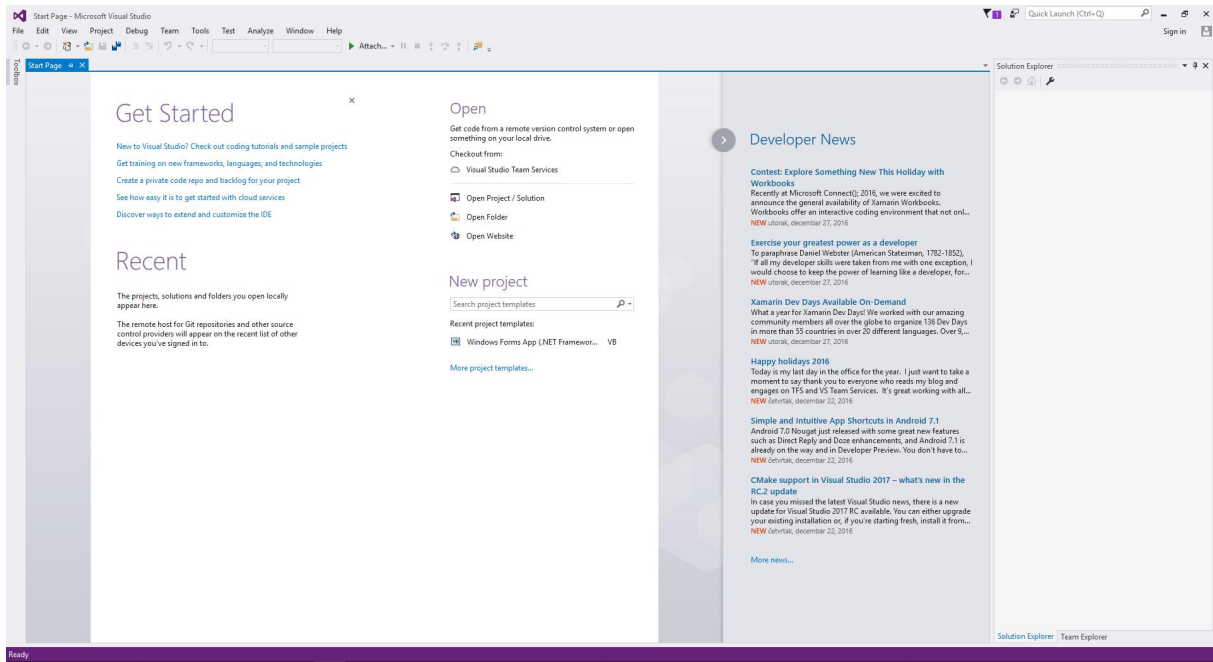


Инсталација је аутоматска и трајаће пар минута у зависности од конфигурације рачунара на који се развојно окружење инсталира. Када се први пут покрене *Visual Studio* захтеваће да се корисник пријави при чему захтева да корисник има *Microsoft* налог. Уколико корисник нема налог, потребно је да га креира, што може учинити кликом на дугме *Sign up* које ће га преусмерити на форму за регистрацију (слика 4).

Слика 4. Креирање *Microsoft* налога

Корисник може изабрати опцију *Not now, maybe later* али ће касније опет бити инсистирано да се корисник аутентификује под истим условима, односно после 30 дана неће моћи више да користи *Visual Studio* док се не региструје. За налог се може користити било који *e-mail* налог, не мора се користити *hotmail/live* налог за регистрацију.

Након пријаве корисника *Visual Studio* ће кренути да учитава подразумевана подешавања. Након учитаних подешавања од корисника се захтева избор теме за *Visual Studio*. По одабиру тему, појавиће се почетни екран – *Start page* (слика 5).

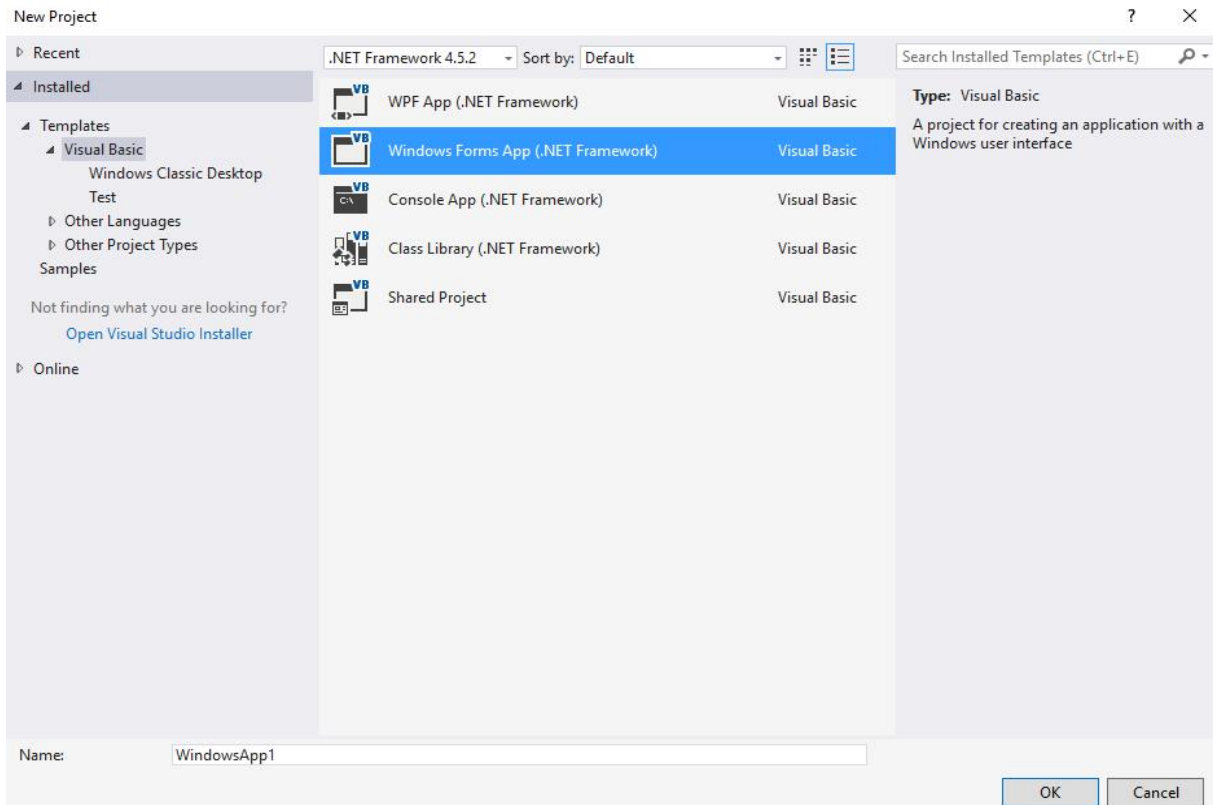


Слика 5. Start page екран приликом покретања Visual Studio

Креирање првог пројекта

Пројекат се може креирати тако што из главног менија одабере опција *File*, а затим у подменију треба изабрати (кликнути на) ставку *New Project...*

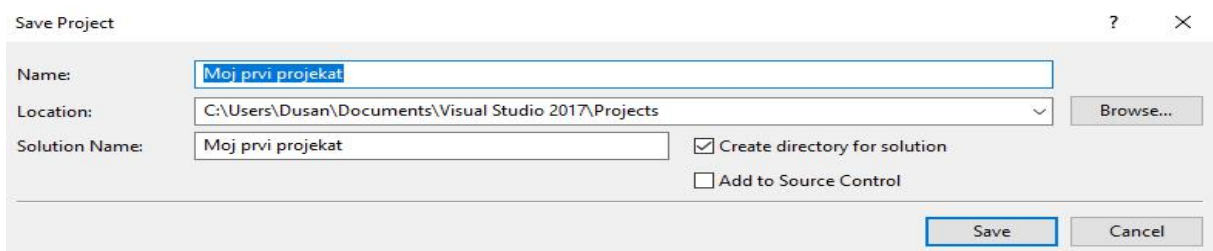
По извршењу ових акција отвара се нови искачући прозор са опцијама за креирање пројекта (слика 6). Са леве стране налазе се програмски језици са којима се може креирати апликације. Са десне стране је тип апликација које могу бити креиране. Из менија са леве стране треба изабрати *Templates* и опцију *Visual Basic* као програмски језик. Из листе са десне стране треба изабрати опцију *Windows Forms App (.NET Framework)*, као на слици 6.



Слика 6. Модални прозор за креирање пројеката

У доњем делу модалног прозора налази се подешавање **Name** које дефинише име пројекта који ће бити креиран. Име пројекта треба да буде смислено и да описује апликацију која ће бити креирана.

У поље поред лабеле *Name* треба унети назив *Moj prvi projekat*. Притиском на дугме **OK** ће бити креиран пројекат *Moj prvi projekat*. Креирани пројекат је сачуван на привременој локацији и потребно га је сачувати на погодној локацији која одговара кориснику што се постиже избором мени ставке *File*, након чега се бира опција *Save* (Ctrl + S). По избору ове опције појављује се прозор на слици 7.



Слика 7. *Save project* дијалог прозор

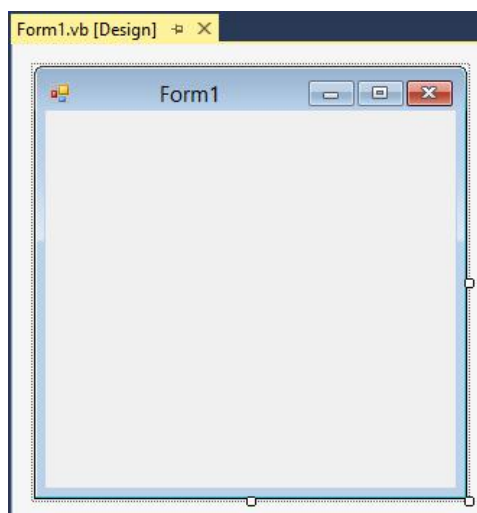
Поред локације за чување пројекта дата је опција *Solution name*. **Solution** је контејнер за пројекте. Некада је потребно да креирана апликација буде састављена из више пројеката, тако да на овај начин они могу бити груписани у целину. За *Solution name* у овом примеру треба навести назив *Uvod u objektno programiranje* и изабрати локацију на којој пројекат треба да буде сачуван.



На локацији где је сачуван *Solution* може се видети да је креиран *.sln* фајл као и директоријум *Moј први пројекат*. *.sln* је екстензија за *Solution* и отвориће у *Visual Studio*-у све пројекте који су груписани у датом решењу.

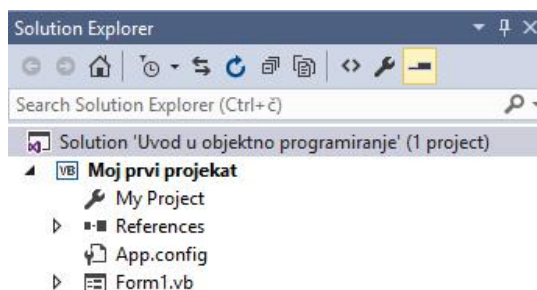
Прозори развојног окружења

Када је креиран пројекат први прозор који се отворена је режим за визуелни дизајн - **Design Mode** (слика 8). У овом режиму, може се визуелно модификовати форма, односно образац.



Слика 8. Режим за визуелни дизајн - *Design Mode*

Са десне стране налази се прозор за рад са контејнерима и пројектима – **Solution Explorer** (слика 9). У оквиру овог прозора налази се цела структура решења. У датом примеру име контејнера је *Uvod u objektno programiranje* а пројекат који је тренутно активан је *Moј први пројекат*. Активни пројекат има свој назив подебљан. Могуће је додати више пројеката у постојеће решење, а поступак ће бити објашњен касније.

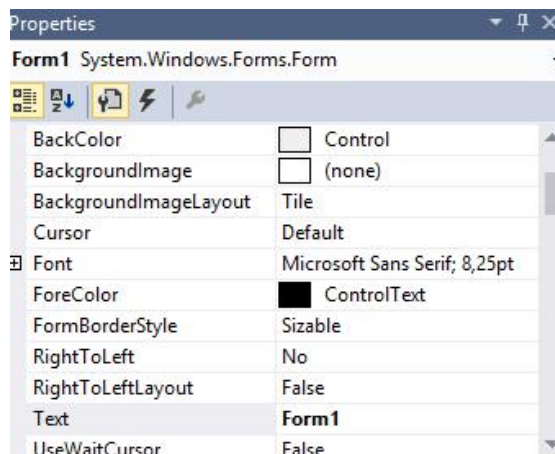


Слика 9. Solution Explorer за контејнер *Uvod u objektno programiranje*

Кликом на форму која се налази у прозору за визуелни дизајн, са десне стране налази се прозор за подешавања – **Properties** (слика 10). У оквиру прозора за подешавања коришћењем визуелног едитора могуће је изменити својства елемента који је изабран. Како је при креирању пројекта једини видљиви



пројекат форма, а други елементи нису додавани, подешавања форме се једино могу мењати у овом примеру.



Слика 10. Прозор за подешавања својстава

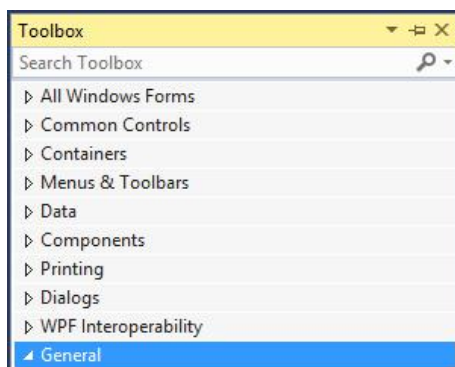
Својства се могу прегледати у два режима – **Alphabetical** () за сортирање својстава алфабетски и **Categorized** () за сортирање својстава по категоријама. Сортирање својстава по категорији је подразумевани преглед.

Са леве стране налазе се три табулатора која отварају додатне прозоре (слика 11). **Server Explorer** везан је за подешавања сервера у случају да креирана апликација преузима податке са неке интернет странице или са локалног сервера. **Data Sources** отвара прозор за подешавања рада са базама података и другим подацима који могу бити извор информација за крерану апликацију.



Слика 11. Табулатори - *Server Explorer*, *Toolbox* и *Data Sources*

Toolbox прозор (слика 12) је за потребе овог курса важан прозор, јер ту се налазе графички објекти који ће бити коришћени за креирање апликација. У оквиру прозора постоје табулатори на чији клик се добијају одговарајуће графичке компоненте. У оквиру табулатора *Common Controls* налазе се објекти који се често користе у апликацијама – лабеле, поља за унос текста, дугмад итд.



Слика 12. Toolbox прозор



Уколико се грешком затвори неки од прозора, може се поново отворити тако што у главној навигацији кликне на ставку *View* а затим изабере жењени прозор.

Покретање апликације

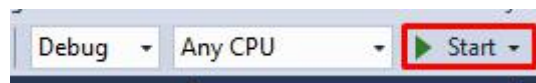
После упознавања са основним прозорима апликације, биће објашњено на који начин се покреће апликација, но пре тога треба се упознати са режимима у којима се апликација може покренути.

Први режим је **Debug** режим који се користити током развоја апликације. Он програмеру дозвољава да тестира функционалност апликације, тражи синтаксне и логичке грешке које могу настати приликом креирања и друге функционалности везане за тестирање софтвера.

Други режим је **Release** режим који служи да апликацију креира као функционалну апликацију спремну за употребу. За разлику од *Debug* режима који креира помоћне датотеке које служе за нпр. праћење грешака, овај режим једноставно пакује библиотеке и податке који су потребни да креирана апликација ради.

Приликом рада на овом курсу препоручујемо *Debug* режим. Покретање апликације се може постићи на више начина:

1. У главном менију у подменију *Debug* изаберити опцију *Start Debugging*.
2. Коришћењем тастатуре, притиском тастера *F5*.
3. У оквиру палете развојног окружења притиском на зелено *Start* дугме као на слици 13.



Слика 13. Покретање апликације коришћењем опције из палете алата

Након ове акције апликација ће бити преведена од стране компајлера интегрисаног у развојно окружење у извршни програм и биће смештена у директоријум *bin* односно у његов поддиректоријум *Debug*. Иако се овде налази *.exe* датотека, не препоручује се коришћење датотеке из датог директоријума, већ је пожељно користити ресурсе из *Release* директоријума.

„Zdravo svete“ апликација



Задатак: Креирати *Windows Forms* апликацију која на екрану има центрирану лабелу на којој стоји текст здраво свете.

За реализацију примера биће коришћен постојећи пројекат *Moј prvi projekat*. Из *Toolbox* прозора потребно је додати објекат *Label*. Он се налази у *Common Tools*. *Toolbox* прозор је подразумевано скривен. Како ће се често користити,

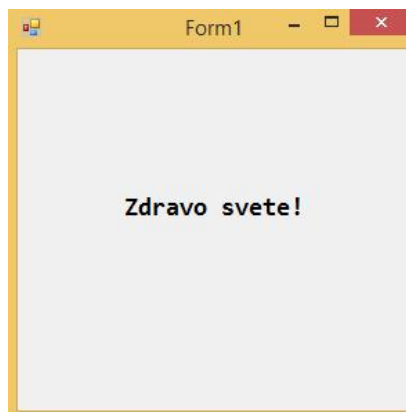


кликом на иконицу  могуће га је закачити тако да стално буде видљив. Додавање објеката на форму може се остварити на два начина:

1. Двоструким кликом на објекат у *Toolbox* прозор након чега ће се објекат појавити у горњем левом углу форме, а затим је могуће *drag and drop*, привлачењем поставити га на жењено место.
2. Притиском левог тастера миша на *Toolbox* прозору, а затим држећи тастер лабела може бити превучен на жељено место на форми.

Превученој лабели потребно је променити одређена својства. У *Properties* прозору треба пронећи својство *Text* и променити текст у *Zdravo svete!*. Могуће је променити и додатна својства на пример величину и тип фонта што се постиже променом својства *Font*.

Пример једног начина израде апликације према захтеву задатка дато је на слици 14.



Слика 14. Пример *Zdravo svete* апликације

У овом поглављу је приказана реализација апликације са једном визуелном компонентом и дат је преглед основних подешавања везаних за креирање контејнера и пројеката. У наредном поглављу биће представљене основне компоненте развојног окружења и први кораци који се тичу интеракције корисника са апликацијом.



БЕЛЕШКЕ

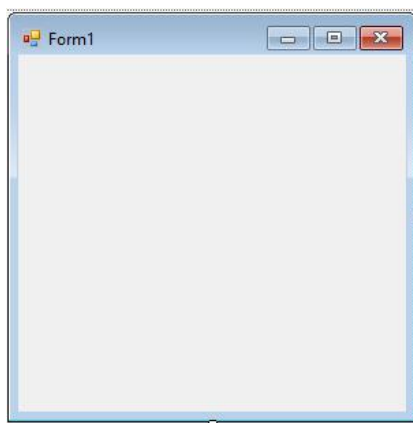


Основни елементи графичког окружења

Једна од предности рада са *Visual Basic* програмским језиком је могућност једноставног развоја апликација са графичким корисничким интерфејсом. У овом поглављу биће представљена четири основна елемента графичког корисничког интерфејса: форма, лабела, поље за унос текста и дугме. Примена осталих елемената, њихова својства и функционалност биће презенована у наредним поглављима.

Форма

Први елемент који се аутоматски креира када креирате *Windows Forms Application* пројекат је форма или образац, (*Form*) (слика 1). Површина елемента је празна без додатних елемената, а из палете са алаткама могуће је превући друге елементе на форму.



Слика 1. Графички елемент форма - *Form*

Форма има заглавље које се састоји од иконице и текста. То су својства *text* и *icon* која се могу променити у палети са својствима. За текст треба унети следећи садржај *Моја прва апликација* а за иконицу узети *ico* датотеку. За претварање било које слике у *ico* датотеку може се користити било који *online* сервис, који се коришћењем *Google*-а или неког другог претраживача може наћи. Кључне речи које треба унети у претраживач су *JPG to ICO* или *PNG to ICO*. За демонстрацију узет је лого школе који је конвертован из слике у иконицу. Резултат измена можете видети на слици 2.

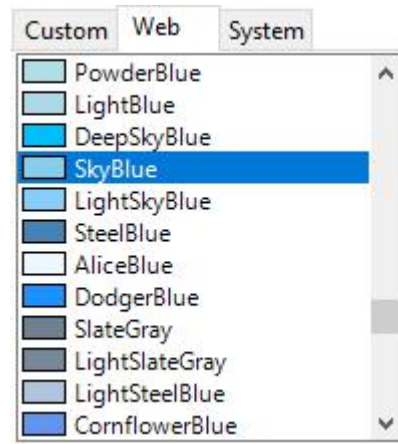


Слика 2. Заглавље Windows Forms апликације

Контејнер форме може да има своју боју позадине, или слику постављену као позадину. Својство за измену боје позадине је **BackColor**. Кликом на опције постоје три могућности избора боја – *Custom*, *Web*, *System*. *Custom* су основне

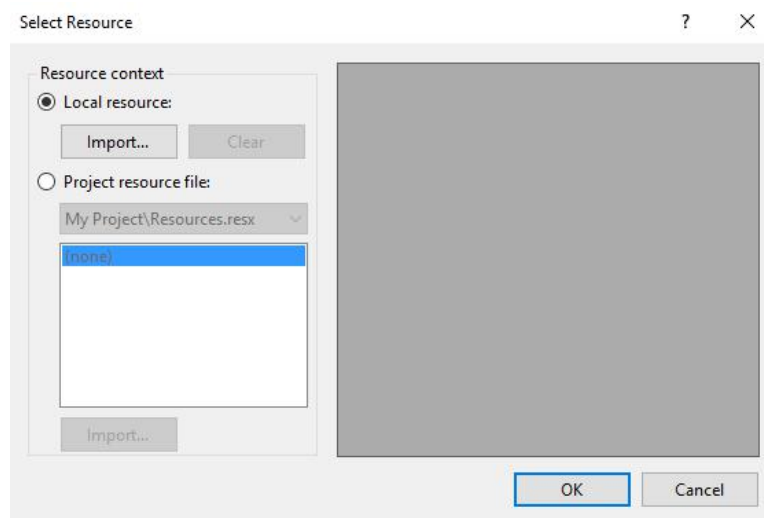


боје, *Web* су боје које се могу наћи у спецификацији веб претраживача, док *System* су предефинисане боје које *Microsoft* користи кроз своје графичке апликације. У *BackColor* својству одаберите боју *SkyBlue* (слика 3).



Слика 3. Палета за избор боја (web)

Поред боје позадине могуће је поставити слику. Слика може заменити позадину или јој бити додаток. Својство за додавање слике позадине је ***BackgroundImage***, а начин приказа слике је ***BackgroundImageLayout***. Коришћење слика или других мултимедијалних садржаја реализује се кроз управљач ресурсима (*Resource Manager*). Могуће је ресурс увести као датотеку са рачунара или креирањем специјалне датотеке (*Resources.resx*) која води евиденцију о ресурсима који се користе у пројекту (слика 4).



Слика 4. Коришћење управљача ресурсима

BackgroundImageLayout има неколико својстава за приказ слике:

None - приказује слику у природној величини почев од горњег левог угла,

Tile - понавља слику као плочице,

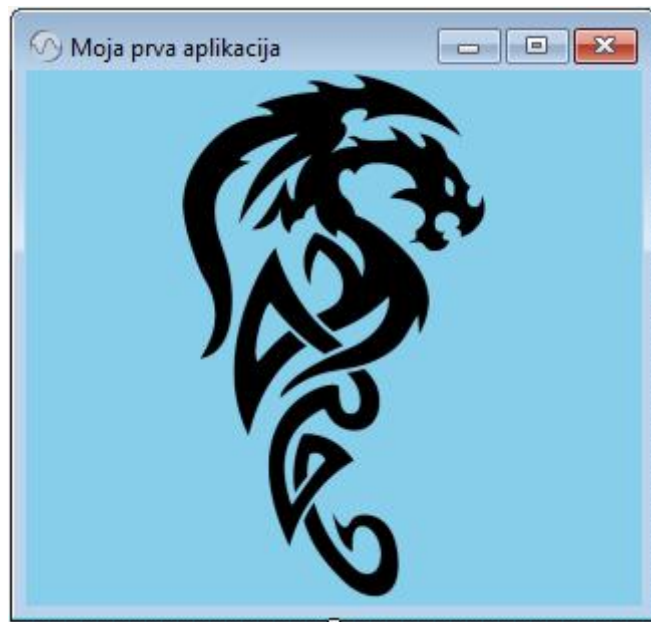
Center – позиционира слику у центар форме (уколико је слика већа од величине форме приказаће само део слике),



Stretch - прилагођава слику (растеже је) да одговара димензијама форме и

Zoom - адаптира слику према величини форме.

Као пример узета је транспарента слика тетоваже змаја а за *BackgroundImageLayout* изабрана је вредност *Zoom*. Изглед форме са овако подешеним параметрима позадине приказан је на слици 5.



Слика 5. Пример коришћења слике за позадину

За измену величине форме користи се својство **Size** које се састоји од два својства *Width* и *Height* (слика 6). Уместо подразумеваних вредности могу се, на пример унети вредности дате на слици 6.

Size	600; 400
Width	600
Height	400

Слика 6. Својство Size и подсвојства - Width и Height

Сва својства форме могуће је изменити у самом коду.

Лабела

Лабела (*Label*) је елемент који се користи за приказ предефинисаног текста. За разлику од поља за унос текста, лабелу се текст може једино изменити писањем кода. Лабела је један од деце елемената форме. Из палете са алатима у секцији *Common Tools* елемент *Label* треба превући на форму. После постављања лабеле на форми она изгледа слично форми датој на слици 7.

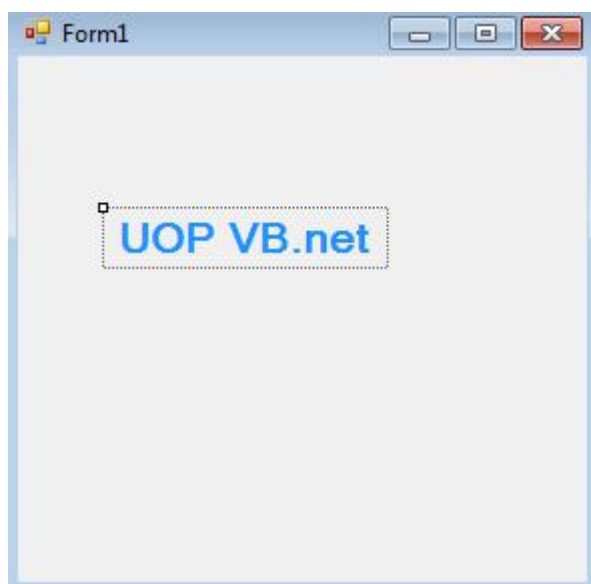


Слика 7. Форма са лабела елементом

Прво својство лабеле које треба изменити је *Text* својство које ће променити текст који се приказује на самој лабели. Уместо *Label1* унети текст у *UOP VB.net*.

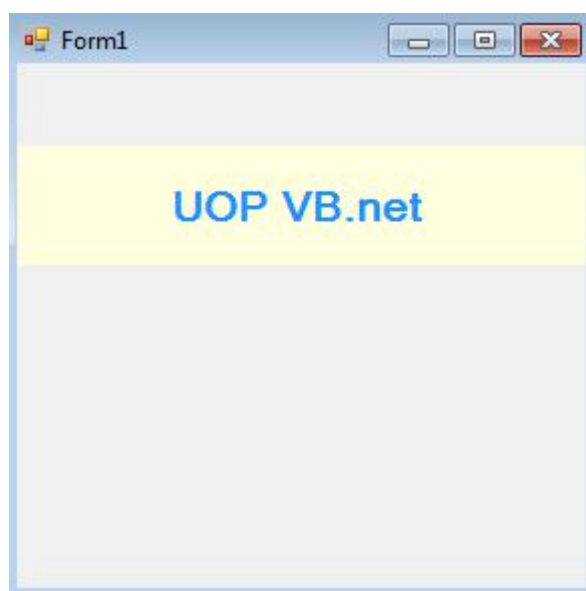
(Name) својство је важно својство и не служи за приказ текста већ за јединствени назив елемента и његову идентификацију. Када се елемент превуче на форму *Visual Studio* му даје подразумевано име које се састоји од имена типа елемента и редног броја елемента. Три превучене лабеле аутоматски добијају имена *Label1*, *Label2*, *Label3*. Име елемента је јединствено. Не могу у истој апликацији да постоје два елемента која имају исту вредност у *(Name)* својству. У сваком тренутку мора бити јасно које име се на који елемент односи јер када је потребно нешто изменити програмским путем навођењем имена тог објекта се врши обраћање елементу са којим је потребно остварити комуникацију тј. извршити неку промену његових својстава или тражити да се обави нека од функционалности коју тај елемент има уграђену или програмирану.

Својство **Font** омогућава да избор породице слова, тип фонта и његову величину. За боју слова текста користи се својство **ForeColor**. Избор палете боја идентичан је оном код форме. На пример, ако би се променила величина на 14px, а за боју фонта из *Web* палете изабрала боја *Dodger Blue* добио би се изглед лабеле као на слици 8.



Слика 8. Предефинисана лабела за корисничка подешавања

Лабели постављеној на форму није могуће променити величину зато што је подразумевано укључено својство *AutoSize*. Ако је потребно променити величину лабеле треба искључити ово својство, тј. изабрати вредност *False*, потом је могуће променити величину лабеле да одговара захтевима задатка тј. потребама корисника. Текст у оквиру контејнера лабеле може се позиционирати помоћу својства лабеле *TextAlign*. На пример својство *AutoSize* постављено је на *False*, својство *TextAlign* постављено је на *MiddleCenter*, а *BackColor* на *LightYellow*. Резултат ових промена приказан је на слици 9.



Слика 9. Лабела са искљученим *AutoSize* својством

Поље за унос текста

Поље за унос текста (*TextBox*) је елемент који дозвољава корисницима да уносе текстуални запис. Налази се у групи елемената секције *Common Tools* и



као и лабела користи се као дете елемент. Има слична својства као и лабела, па ће у наставку бити речи о својствима која су јединствена за овај елемент.

Multiline својство одређује да ли ће бити једнолинијских или вишелинијских корисничких уноса. Уколико је вредност *True* унос је вишелинијски, уколико не унос је једнолинијски. **Lines** својство нам омогућава да се унесе предефинисани вишелинијски текст ако је **Multiline** својство активно. У супротном свака линија ће бити спојена на претходну.

ReadOnly својство онемогућава корисника да унесе текст у поље, текст је само за читање.

Scrollbars својство служи за додавање хоризонталног и/или вертикалног клизача. Ово својство се користи када је својство **Multiline** активно.

Контејнер за слику

Контејнер за слику (*PictureBox*) служи за додавање слике у апликацију. Од својстава која су значајна, три су везана за сам ресурс слике: *Initial image*, *Image* и *Error Image*. **Initial image** је иницијална слика која се приказује док се права слика учитава. Препорука је да се за ову слику користи *GIF* слика *spinner*-а (слика 10).

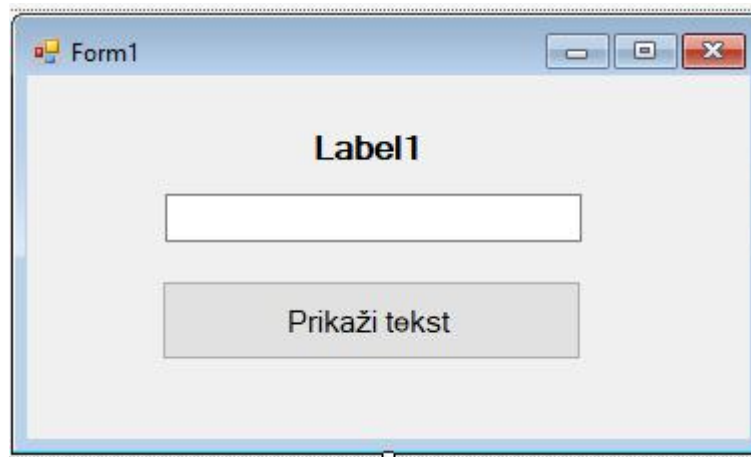


Слика 10. Пример *spinner*-а за *Initial image* својство

Својство **Image** представља слику коју треба приказати, док својство **Error Image** приказује слику која се отвара када жељена слика не може да се прочита.

Догађаји и динамичка измена својстава

У претходном тексту је описано како се коришћењем *Properties* прозора могу мењати својства, међутим, често је потребно да се у току извршавања програма промени неко својство графичког елемента то се постиже писањем кода. Нека је дата форма приказана на слици 11.



Слика 11. Пример форме са три графичка елемента

У циљу реализације комуникације између корисника и апликације битно је знати како функционише ток извршања *event-driven* апликација. **Event-driven** апликације су апликације где се интеракција реализује извршавањем догађаја. Сваки догађај дефинисан је следећим током:

1. **Корисник покрене догађај над апликацијом или апликација креира сама догађај.** Догађај који корисник може да покрене над апликацијом је на пример клик на дугме или прелаз миша преко лабеле. Догађај који апликација може да изазове или креира над собом је на пример измена података када се апликација покрене или извршавање неке радње у задатом временском интервалу.
2. **Догађај утиче на елемент/е.** На пример када корисник кликне на дугме, лабела промени боју. Такође је могуће да се вредност својства једног елемента додели својству другог елемента. На пример из поља за унос текста, унети текст корисника доделимо другом елементу на пример лабели.
3. **Корисник добија неку врсту повратне информације.** Корисник види визуелно измене или добија неку врсту поруке о томе да је догађај успешно реализован.

Именовање догађаја и елемената

У наведеном току извршавања, догађај мора бити прецизно дефинисан. Да би се догађај реализовао онако како је програмер замислио, потребно је знати јасне одговоре на питања:

1. Који је елемент захтевао догађај?
2. Који је догађај позван?
3. Који је елемент (ако се извршавају промене над елементом) измењен. Који је исход написаног кода?
4. Како корисника обавестити о изменама насталим позивањем догађаја?



Управо одговори на претходно наведена питања захтевају давање јединствених имена елементима и догађајима, па би било добро водити рачуна о следећем:

1. Када се елемент превуче на форму он добија подразумевано име. За дугме то ће бити *Button1*, следеће дугме биће *Button2* итд., за лабелу *Label1*, *Label2*, за поље за унос текста *TextBox1*, *TextBox2*. Проблем овог именовања које подразумевано даје *Visual Studio* је да на форми може постојати више десетина елемената који су дугмад или лабеле или поља за унос текста и са подразумеваним именима је тешко наћи жењени елемент, поготово и сложеним апликацијама.
2. Елементима треба дати смислено име које га описује. То име треба да прати следећа правила:
 - Име елемента мора бити јединствено и не може на истој форми да постоји други елемент или догађај са датим именом.
 - Име елемента мора да садржи само слова и бројеве. Име не може да има размак или друге специјалне знакове.
 - Име елемента не може да почиње са бројем. Име елемента мора почети са малим или великом словом.
 - Уколико је елемент графички елемент, препорука је да се испред елемента поставе одговарајући префикси. Листа префикса за неке учестале елементе дата је у табели 1.

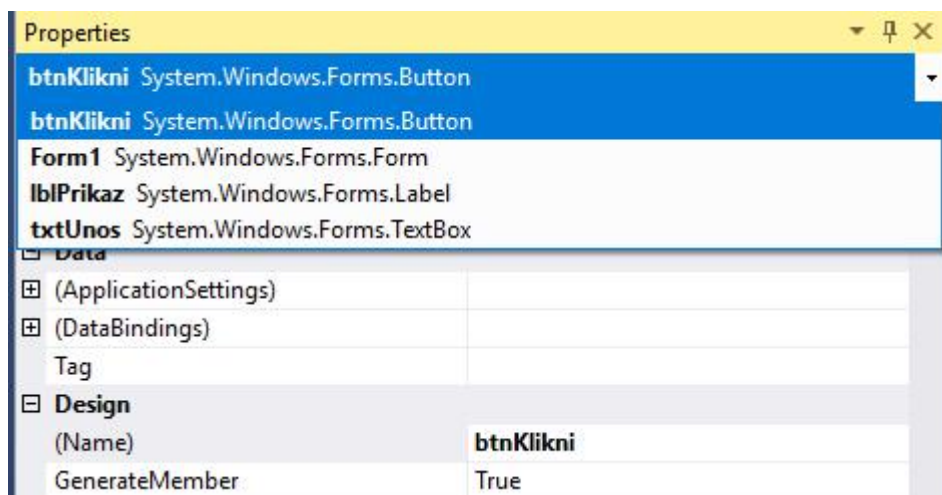
Графички елемент	Префикс	Пример
Button	btn	btnOpen
Check box	chk	chkReadOnly
Combo box, drop-down list box	cbo	cboEnglish
Command button	cmd	cmdExit
Common dialog	dlg	dlgFileOpen
Date picker	dtp	dtpPublished
Form	frm	frmEntry
Frame	fra	fraLanguage
Image	img	imgIcon
ImageList	ils	ilsAllIcons
Label	lbl	lblHelpMessage
List box	lst	lstPolicyCodes
Menu	mnu	mnuFileOpen
Option button	opt	optGender
Picture box	pic	picVGA
ProgressBar	prg	prgLoadFile
RichTextBox	rtf	rtfReport



StatusBar	sta	staDateTime
TabStrip	tab	tabOptions
Text box	txt	txtLastName
Timer	tmr	tmrAlarm

Табела 1. Учестали префикси и примери за именовање елемената

У претходно приказаном примеру. Лабели са текстом *Label1* према конвенцији треба дати име *lblPrikaz*, пољу за унос текста *txtUnos* а дугмету *btnKlikni*. У *Properties* прозору потребно је изменити (*Name*) својство. У падајућој листи испод *Properties* заглавља могу се видети новоименовани елементи (слика 12).



Слика 12. Падајућа листа са новоименованим елементима

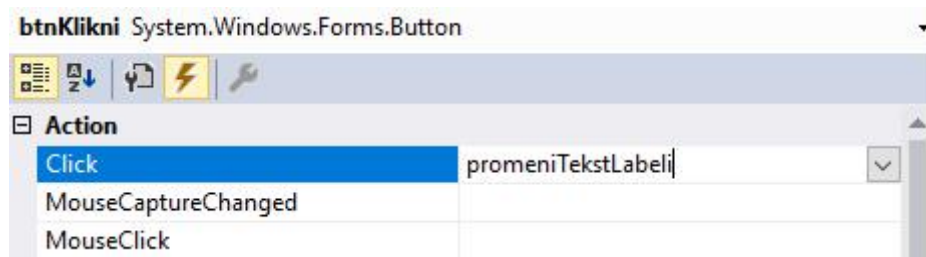
Дефинисање основних догађаја



Задатак: Креирати форму са слике 11. Када корисник унесе текст у поље за унос текста, кликом на дугме потребно је да се текст у лабели са текстом *Label1* замени са текстом који је унео корисник.

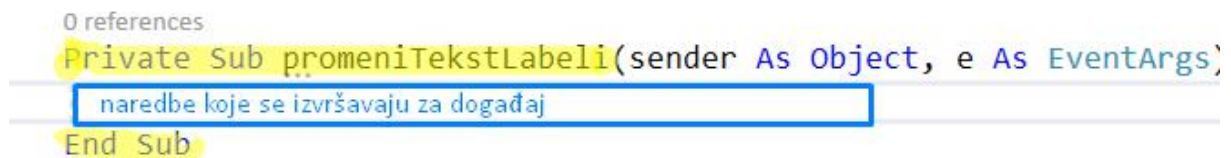
Из поставке задатка је познат одговор на питање „Који је елемент позвао догађај?“ то дугме на које је корисник кликнуо - *btnKlikni*. Одговор на питање „Који је догађај позван?“. Позван је догађај клик дугмета, који програмер треба да напише.

У *Properties* прозору налази се иконица за догађаје (⚡). Кликом на њу појавиће се листа догађаја које елемент може реалзовати. Догађај који је потребно креирати је везан за акцију *Click*. Из листе понуђених догађаја за дугме треба изабрати овај догађај и попунити поље десно од имена догађаја које представља назив догађаја. Догађаје је пожељно смислено именовати тако да описују радњу коју догађај треба да уради. За именовање догађаја важе иста правила као и за именовање графичких елемената. Такође, догађај и било који графички елемент на форми не смеју имати исто име. Име догађаја у решењу постављеног задатка биће *promeniTekstLabeli* (слика 13).



Слика 13. Давање имена догађаја за клик акцију дугмета „btnKlikni“

Отвара се едитор за уређивање кода (*Code Editor*) са иницијалним кодом за догађај. Може се приметити да овом догађају није дата никаква функционалност, што се може видети на слици 14, где је жутом бојом обележен код који представља почетак и крај догађаја. Између ових линија програмер пише делове кода којим реализује жељени догађај.



Слика 14. Код генерисан за догађај „promeniTekstLabeli“

Следећи корак је написати кода који кореспондира одговору. Елемент који ће бити измењен је *lblPrikaz* а елемент од ког ће бити узета вредност за измену је *txtUnos*. Кориснику није потребно приказати обавештење о измени јер ће визуелна промена бити индикатор да се догађај реализовао.

Када је потребно приступити неком својству објекта користи се следећа синтакса: **име објекта.име својства**. На пример *Text* својству елемента *lblPrikaz* приступа се писањем *lblPrikaz.Text*. Приступање својству боје фонта - *lblPrikaz.ForeColor*.

У демонстрационом примеру потребно је променити *Text* својство лабеле *lblPrikaz* тако што ће бити додењена вредност *Text* својства поља за унос текста *txtUnos*. Када је потребно изменити својство, односно доделити му неку вредност мора се користити **оператор доделе** знак = (једнако). Наредба за доделу има формат **име објекта.име својства = вредност**. Вредност може бити или својство од неког другог објекта или директна вредност у коду (хардкодована, *hardcoded*).

Решење примера је дато следећим кодом.

1. `Public Sub promeniTekstLabeli(sender As Object, e As EventArgs)`
`Handles btnKlikni.Click`
2. `lblPrikaz.Text = txtUnos.Text`
3. `End Sub`

Комбинацијом тастера *Ctrl* и *F5* покреће се креирана апликација.



Корисно је напоменути да је за повратак у режим за дизајн пречица на тастатури комбинаца тастера *Shift* и *F7*. За прелазак из режима дизајна назад у едитор користи се тастер *F7*.



БЕЛЕШКЕ



Рад са променљивама

Променљиве служе за складиштење података и омогућавају да се њихове вредности могу по потреби користити у програму кад год је то потребно. Давање смисленог имена променљивама је добра програмерска пракса, јер се тиме побољшава читљивост програма, што је јако битно ако је потребно додати или изменити неку функционалност програма.

На слици 1 дат је приказ једног сегмента меморијског простора рачунара. Са леве стране дате су адресе у меморији а са десне стране бинарни запис података.

	...	
37FD00	01001100	
37FD01	1001010	
37FD02	00100110	
37FD03	01101101	
	...	

Слика 1. Пример чувања података у меморији рачунара

Да би програм користио неки податак из меморије рачунара неопходно је да је позната његова локација у меморији рачунара тј. адреса, величина меморије коју он заузима и интерпретација, што подразумева трансформацију из бинарног записа у податак који је разумљив за корисника (текст, слика, видео, ...). Један од проблема је и што неки подаци који имају сложену структуру и у меморији рачунара не морају бити смештени на једној адреси већ у зависности од величине податка они могу бити подељени на више делова и чувани на више адреса које не морају да буду узастопне.

Оперативни систем рачунара, када се на њему покрене програм, додељује део меморије програму који он може да користи. Оперативни систем мора добити од самог програма информацију колико меморијског простора је потребно за складиштење неког податка, као и колика су потраживања програма у целости. Меморија коју програм добије је проширива али основна меморија се добија на основу садржаја – колико је потребно издвојити простора за чување текста, графичког приказа, да ли постоји интеракција итд. Процес је сложен, али оно што битно је истаћи је да програмер не одлучује о додели меморијских адреса већ оперативни систем. Програмер треба да води рачуна да обезбеди рационално коришћење меморије (memory leak). Како остварити овај захтев? Један од начина је писање програм који ће захтевати коришћење ресурса који су неопходни и ослобађање истих кад нема више потребе за њима и при том треба водити рачуна да корисник који инсталира програм буде свестан минималних захтева у погледу перформанси рачунара на коју се програм инсталира.



Улога променљивих у коришћењу ресурса је вишеструка. Оне омогућавају програмеру да користи потребне податке, а да при том не задаје и не зна тачну њихову адресу у меморије и омогућују му да одреди тип податка који ће бити смештен у променљивој. Познавање типа податка је важно јер на основу типа податка програм зна колико меморијског простора треба да тражи од оперативног система за смештање променљиве и како да садржај смештен на тој локацији интерпретира.

Наредни пример из реалног живота може приближити читаоцу појам променљиве. Приликом селидбе из једног стана или куће у стан или кућу на другој локацији потребно је свари пренети са једног места на друго. Један сценарио би био да се једноставно смести што више ствари у камион за селидбу и да се ствари на новој локацији истоваре. Проблем је што су неке ствари ломљиве, неке нису ни приоритетне а мало теже је ствари ставити на ново место са овим методом селидбе. Други сценарио, логичнији је да се ствари сортирају по неком критеријуму и да се упакују у различите кутије. Ове кутије би било корисно уредно обележити. На овај начин и ако није све распаковано, лакше се може наћи нека ствар и распаковање је лакше, јер се тачно узимају ствари које су потребне у датом тренутку.

Свака од кутија је променљива, обележавање кутије је њено име, а ствари које се налазе у кутију су њена вредност. На слици 2 дате су неке кутије.



Слика 2. Пример кутија са називом

Кутије у примеру имају имена категорије предмета које су у њима. Ове категорије би могле бити тип податка. Код рачунара име променљиве мора да се разликује од имена тип податка, јер сваки програмски језик има резервисане речи. Оне заједно са правилима за писање програма представљају синтаксу програмског језика. Ако би име променљиве и име податка били исти у том случају преводилац не би знао да ли се мисли на тип податка или на променљиву.

Типови података у Visual Basic-у дати су у табели 1.

Тип податка	Величина у меморији	Опсег вредности
Boolean	У зависности од	True или False



	верзије оперативног система	
Byte	1 byte	0 до 255 (неозначени)
Char (један карактер)	2 bytes	0 до 65535 (неозначени)
Date	8 bytes	0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999
Decimal	16 bytes	0 до +/- 79,228,162,514,264,337,593,543,950,335 (+/- 7.9...E+28) [†] без децималне тачке; 0 до +/-7.9228162514264337593543950335 са 28 места после децималне тачке
Double	8 bytes	-1.79769313486231570E+308 до - 4.94065645841246544E-324 [†] за негативне вредности; 4.94065645841246544E-324 до 1.79769313486231570E+308 [†] за позитивне вредности
Integer	4 bytes	-2,147,483,648 до 2,147,483,647 (означени)
Long (long integer)	8 bytes	-9,223,372,036,854,775,808 до 9,223,372,036,854,775,807 (9.2...E+18 [†]) (означени)
Object	4 bytes на 32-bit систему 8 bytes на 64-bit систему	Било који тип се може сместити у тип <code>Object</code>
SByte	1 byte	-128 до 127 (означени)
Short (short integer)	2 bytes	-32,768 до 32,767 (означени)
Single (single- precision floating- point)	4 bytes	-3.4028235E+38 до -1.401298E-45 [†] за негативне вредности; 1.401298E-45 до 3.4028235E+38 [†] за позитивне вредности



String (variable-length)	У зависности од верзије оперативног система	0 до приближно 2 билиона Unicode карактера
UInteger	4 bytes	0 до 4,294,967,295 (неозначени)
ULong	8 bytes	0 до 18,446,744,073,709,551,615 (1.8...E+19 †) (неозначени)
User-Defined (structure)	У зависности од верзије оперативног система	Сваки објекат структуре је дефинисан својим атрибутима и меморијски заузима суму простора меморије сваког атрибута.
UShort	2 bytes	0 до 65,535 (unsigned)

Табела 1. Типови података у Visual Basic-у

Типови података говоре рачунару колико је меморије потребно за складиштење променљиве, на који ће начин бити интерпретиран садржај те меморије (да ли су у питању текстуални подаци или бројеви, ако су бројеви да ли су цели или не...) и које се операције над тим подацима могу извршити. На пример нека је на празном папиру написано „2017“. Овај број би могао схваћен на више начина, на пример као један целобројни четвороцифрени број, или као ознака године, или да су то четири цифре једна за другом итд. Тачност тог конкретног навођења броја добија се тек њеним стављањем у неки конкретан контекст. Исто ово важи и за рачунар. Програмер типом податка даје упутство рачунару како да податак „2017“ смести и које операције су могуће над тим податком.

Декларација и иницијализација променљивих

Декларација служи за одређивање типа променљиве и издвајање меморијског простора за променљиву. Пре првог коришћења променљиве она мора бити декларисана.

Иницијализација је почетна додела вредности променљивој.

Декларација променљивих

Декларација променљиве се пише **само једном**. Иницијализацијом се сматра само прва додела вредности променљивој, а вредност променљиве може у програму да буде измењена произвољан број пута.

У Visual Basic-у се декларација променљиве врши на следећи начин:

Dim **Име променљиве** **As** **Тип податка**

„Dim“ је резервисана реч која је скраћеница од „dimension“ односно говори преводиоцу да треба да издвоји меморијски простор за променљиву. „As“ је



резервисана реч која говори који ће тип податка ће бити сачуван у променљивој. Типови података који могу да се користе наведени су у табели 1.

При избору имена променљиве, као и процедуре, класе, константе и аргумената морају се поштовати правила за именовање која је прописао Microsoft:

- Први карактер мора бити слово.
- Не може се користити белина (space, enter), тачка (.), узвичник (!) или карактери @, &, \$, #. Такође је пожељно да се користи енглески алфабет за слова.
- Име не може да буде дуже од 255 карактера.
- Не може се именовати као нека од резервисаних речи у Visual Basic-у.

У табели 2 налазе се примери исправних и неисправних имена променљивих:

✓	Pera	✓	autobus
✓	pi314	✗	3ci
✗	marko@gmail.com	✗	ime.prezime
✗	Ime_prezime	✓	ImePrezime
✗	Upozorenje!	✗	#trend

Табела 2. Правилна и неправилна имена променљивих



Задатак: Декларисати променљиве у Visual Basic-у са следећим карактеристикама:

- Име је „mojBroj“ а тип податка је *Integer*
- Име је „operand2“ а тип податка је *Double*
- Име је „pozdravnaPoruka“ а тип податка је *String*
- Име је „peraperic“ а тип податка је *Osoba* (ово је неки тип податка који је програмер дефинисао пре ове декларације -User-Defined)

Код за декларације дат је у наставку:

1. `Dim mojBroj As Integer`
2. `Dim operand2 As Double`
3. `Dim pozdravnaPoruka As String`
4. `Dim peraperic As Osoba`

Иницијализација променљивих

Иницијализација променљиве је почетна додела вредности која одговара њеном типу податка. Иницијализација се реализује коришћењем оператора



једнако (=). Оператор = је наредба доделе. Она се реализује тако што се у меморијску локацију променљиве са леве стране оператора = уписује садржај променљиве или константе са десне стране или вредност израза са десне стране, ако је у десном делу наредбе израз. У наставку следе примери иницијализације за неке од типова података из претходног примера:

1. `Dim mojBroj As Integer`
2. `mojBroj = 5`
3. `Dim operand2 As Double`
4. `operand2 = 3.75`
5. `Dim pozdravnaPoruka As String`
6. `pozdravnaPoruka = "Zdravo svete"`

О сложеним типовима података (Osoba) биће речи у каснијим поглављима. Постоје разлике како се подаци додељују. За нумеричке и *boolean* вредности довољно је само задати вредност која се смешта, док за *Char* и *String* вредност мора да се налази под наводницима. Код *Char* користи се апостроф (') а код *String* користе се наводници (").

Иницијализација се може реализовати у истој линији кода као и декларација:

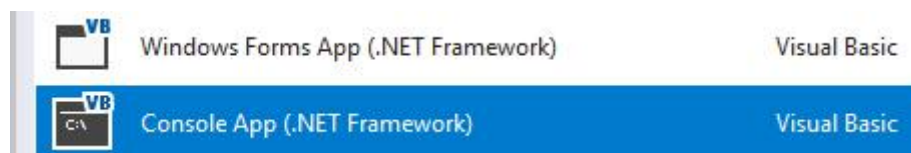
1. `Dim mojBroj As Integer = 5`
2. `Dim operand2 As Double = 3.75`
3. `Dim pozdravnaPoruka As String = "Zdravo svete"`

Конзолна апликација



Задатак: Написати програм који од корисника тражи да унесе текст а затим тај текст исписује у конзоли уз одговарајућу поруку.

Овај задатак биће решен конзолном апликацијом. Да би креирали пројекат за конзолну апликацију приликом прављења новог пројекта потребно је изабрати „Console App (.NET Framework)“ (слика 3).



Слика 3. Креирање пројекта за конзолну апликацију

Када је креиран конзолни пројекат отвориће се едитор за писање кода са почетним кодом за апликацију. Треба унети следећи код:

1. `Module Module1`
2. `Sub Main()`

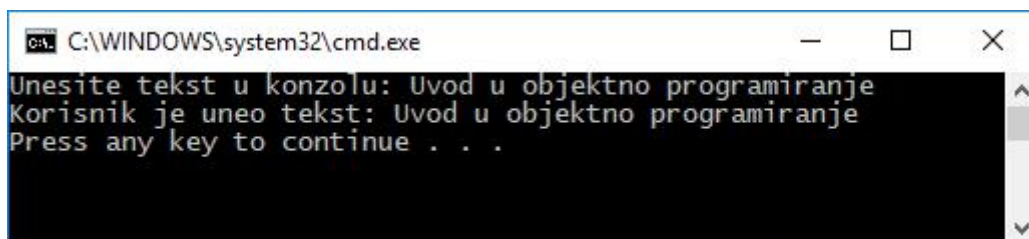


```
3.         Dim unetiTekst As String
4.         Console.WriteLine("Unesite tekst u konzolu: ")
5.         unetiTekst = Console.ReadLine()
6.         Console.WriteLine("Korisnik je uneo tekst: " &
    unetiTekst)
7.     End Sub
8. End Module
```

„Module Module1“ и његов крај *End Module* су подразумевани контејнер из ког се покреће апликација, а *Sub Main()* и њен крај *End Sub* су контејнер за писање програма. Детаљније о овом генерисаном коду биће речи у наредним поглављима приручника. У линији 3 декларисана је променљиву *unetiTekst* која ће сачувати вредност коју је унео корисник. У 4 линији позивана је *Console.WriteLine* метода (функција) која служи да испише текст који задајемо. Када се текст задаје променљивама или методама, он се мора се навести под наводницима.

У 5. реду се врши иницијализација променљиве *unesiTekst*. Неке методе могу да врате вредности, једна од таквих метода је *Console.ReadLine*. Ова метода ће паузирати извршавање програма и чекаће од корисника да унесе текст који ће вратити као вредност. Ту вредност која је враћена треба сместити у неку променљиву иначе се дати унос губи (неће бити исписан на екран, а како нигде није сачуван програм ће наставити нормално извршавање као да се тај код није ни извршио).

У 6. реду се извршава исписивање корисничког уноса. Прво се исписује предефинисан текст „Korisnik je uneo tekst: “ а затим коришћењем оператора надовезивања (&) из променљиве *unetiTekst* се узима унета вредност и надовезује се на предефинисани текст. Резултат примера извршавање апликације дат је на слици 4.



Слика 4. Извршавање конзолне апликације са корисничким уносом

Аритметички оператори и конверзије вредности

За нумеричке типове података постоје одређени оператори за операције сабирања, множења, одузимања и дељења. Ти аритметички оператори имају ознаке плус (+), звезда (*), минус (-) и коса црта (/) респективно. Од додатних оператора ту је оператор степеновања ознаке капа (^) и оператор за остатак при дељењу ознаке *mod*.



С обзиром да апликације у овом приручнику углавном користе податке које уноси корисник, неопходно је нагласити да сваки кориснички унос прочитан из поља за унос је текстуалног типа. Без обзира да ли је корисник унео „ааа“ или „3.27“ оба уноса су текстуалног типа. Аритметички оператори не могу да раде са текстуалним подацима па је унети податак потребно претворити у нумерички тип. У Visual Basic-у постоји функција која конвертује свој аргумент, који је текстуалног типа у одговарајући нумерички тип податка – **Val**.



Задатак: Написати конзоларни програм који симулира калкулатор.

Од корисника се тражи да унесе два броја (могу бити реални или целобројни) и за унете бројеве кориснику треба исписати резултате сабирања, одузимања, множења и дељења.

Потребно је да прво декларисати две променљиве на пример *prviBroj* и *drugiBroj* које ће бити типа *Double*. Зашто *Double*? *Double* представља скуп реалних бројева, а цели бројеви су део тог подскупа, тако да иако корисник унесе на пример вредности 5 и 4, оне ће се исто третирати као реални бројеви.

За почетак треба декларисати променљиве као у наставку:

1. `Sub Main()`
2. `Dim prviBroj As Double`
3. `Dim drugiBroj As Double`
4. `End Sub`

Сада пре *End Sub* треба додати интеракцију са корисником. Користи се метода *Console.ReadLine* која чита унос корисника, који је текстуалног типа. Овај текст треба да претворити у нумеричку вредност:

1. `Console.WriteLine("Unesite prvi broj: ")`
2. `prviBroj = Val(Console.ReadLine())`
3. `Console.WriteLine("Unesite drugi broj: ")`
4. `drugiBroj = Val(Console.ReadLine())`

Са функцијом *Val* корисников унос се из текстуалног записа претвара у одговарајући тип податка (*Double*). Преостало је још да се изврше одговарајуће операције над бројевима и да се њихов резултат испише. Додати следеће код:

1. `Console.WriteLine("Zbir " & Str(prviBroj) & " i " & Str(drugiBroj) & " je" & Str(prviBroj + drugiBroj))`
2. `Console.WriteLine("Razlika " & Str(prviBroj) & " i " & Str(drugiBroj) & " je" & Str(prviBroj - drugiBroj))`
3. `Console.WriteLine("Proizvod " & Str(prviBroj) & " i " & Str(drugiBroj) & " je" & Str(prviBroj * drugiBroj))`



4. `Console.WriteLine("Količnik " & Str(prviBroj) & " i " & Str(drugiBroj) & " je" & Str(prviBroj / drugiBroj))`

Оператором `&` је већ помињан у претходном примеру и речено је да он служи за спајање текста. Како ни бројеви ни њихов резултат нису текстуалног типа, треба их претворити у текст коришћењем функције `Str`. У оквиру функције су коришћени директно оператори за сабирање, одузимање, множење и дељење. Ово је могуће зато што као и у математици, заграде имају највећи приоритет па се прво израчунава вредност операције, а затим се функцији прослеђује израчуната вредност коју она конвертује у текст. Исход једног извршавања програма је дат на слици 5.

```
Unesite prvi broj: 6
Unesite drugi broj: 3
Zbir 6 i 3 je 9
Razlika 6 i 3 je 3
Proizvod 6 i 3 je 18
Kolicnik 6 i 3 je 2
Press any key to continue . . .
```

Слика 5. Извршавање примера за симулацију калкулатора

Приликом спајања два *String* –а може се користити и оператор `+` уместо оператора `&`. Дакле, оператор `+` се користи за сабирање бројних вредности али и за надовезивање теста.

Програм за испис текста који је коришћен у овом примеру је робустан. Користи оператор за спајање, функцију за претварање броја у текст и није најчитљивији. *Microsoft* је препознао овај проблем и почев од стандарда језика 2015, увео је нови начин за испис текста који би требао да смањи потребу за коришћење оператора спајања. Нова синтакса за испис почиње са доларом и наводником `$` и завршава се са наводником `"`. Између њих наводи се текст за испит. Уколико је део тог текста нека вредност, позив функције или било који други тип израза, он се ставља у витичасте заграде `{}`. У наставку дат је исти пример исписа текста за тражени задатак.

1. `Console.WriteLine($"Zbir {prviBroj} i {drugiBroj} je {prviBroj + drugiBroj}")`
2. `Console.WriteLine($"Razlika {prviBroj} i {drugiBroj} je {prviBroj - drugiBroj}")`
3. `Console.WriteLine($"Proizvod {prviBroj} i {drugiBroj} je {prviBroj * drugiBroj}")`
4. `Console.WriteLine($"Količnik {prviBroj} i {drugiBroj} je {prviBroj / drugiBroj}")`

У овом приручнику најчешће се примењује овај начин исписа текста у комбинацији са оператором спајања.



Опсег видљивости променљивих – локалне и атрибутске

Променљиве се у програму могу декларисати по потреби на разним местима. У наредним поглављима биће детаљно описане све врсте променљивих у односу на то где су дефинисане и у ком делу програма важе. Када је реч о апликацијама које су до сад разматране, постоје два типа када говоримо о апликацији са формом. Први тип је атрибутска променљива односно **атрибут**, а други тип променљиве је **локална** променљива.

Локална променљива се декларише унутар догађаја. Локалне променљиве су привремене променљиве и служе када је потребно сачувати вредност за неки међукорак. Локалне променљиве важе само у датом догађају. Више догађаја може имати локалну променљиву истог имена (слика 6). Локалне променљиве истог назива не морају да буду ни истог типа и не односе се на исти меморијски простор.

Атрибут се декларише испод самог назива класе и видљив је у свим догађајима (слика 6). Ово својство омогућава да се одређена вредност чува или примени касније током извршавања програма. На пример у Windows Calculator апликацији, када се израчуна неки израз (нпр. сабирање два броја), његова вредност се може користити у неком изразу који следи.

```
Public Class Form1
    Dim atributska As Integer

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim lokalna As Integer

    End Sub

    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim lokalna As String

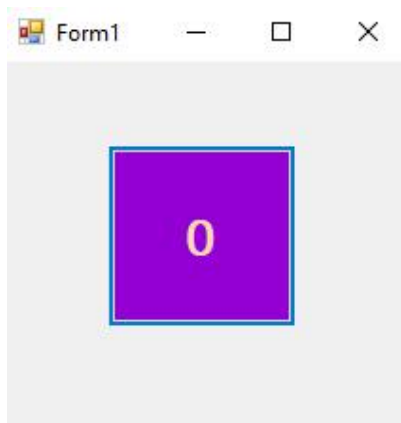
    End Sub
End Class
```

Слика 6. Опсег видљивости променљивих - атрибут и локалне



Задатак: Написати програм који броји колико је пута корисник кликнуо на дугме. Број треба приказати у тексту дугмета. Почетни текст у дугмету је „0“.

Овим примером ће бити приказана разлика између употребе атрибутских и локалних променљивих. Креирати дизајна као на слици 7.



Слика 7. Пример дизајна

Дугмету дати назив *btnKlik*, а догађај *inkrementirajKlik*. Додати код:

1. `Public Class Form1`
2. `Private Sub inkrementirajKlik(sender As Object, e As EventArgs) Handles btnKlik.Click`
3. `Dim brojKlikova As Integer = 0`
4. `brojKlikova = brojKlikova + 1`
5. `btnKlik.Text = Str(brojKlikova)`
6. `End Sub`
7. `End Class`

У 3. реду је декларисана променљива која броји кликове и постављена јој је вредност на нулу, у четвртом реду се вредност променљиве *brojKlikova* повећава за 1 тако што се вредност која је претходно била смештена у овој променљивој повећа за 1 и ова увећана вредност се смешта уместо старе вредности. У 5. Реду се нова вредност променљиве конвертује у текст и приказује се у тексту дугмета.

Када се покрене ова апликација она не ради онако како је очекивано, већ увек показује вредност 1. Зашто? У 3. реду претходно написаног програма је проблем. Догађај се извршава сваки пут када корисник кликне на дугме, и низ наредби у датом догађају се такође поново извршава сваки пут. То укључује и декларацију и иницијализацију променљиве *brojKlikova*. Сваки пут када корисник кликне на дугме, вредност се поставља на нулу а затим повећа за 1, тако да на дугмету ће увек писати вредност 1 после сваког клика.

Како исправити грешку? Променљива *brojKlikova* треба да се постави као атрибутска променљива. Кориговати претходни програм на следећи начин:

1. `Public Class Form1`
2. `Dim brojKlikova As Integer = 0`



3. `Private Sub` inkrementirajKlik(sender As Object, e As EventArgs) Handles btnKlik.Click
4. `brojKlikova = brojKlikova + 1`
5. `btnKlik.Text = Str(brojKlikova)`
6. `End Sub`
7. `End Class`

Променљива *brojKlikova* је издвојена изван догађаја и када се покрене апликација добијаће се очекивани резултат тј. може се пратити колико је пута корисник кликнуо на дугме.

Math класа

Једна од предности језика високог нивоа је што постоје одређене библиотеке које омогућују програмеру да користи већ постојеће функције које су написали тимови програмера који су развили тај програмски језик, уместо да сваки пут сам пише те функције. *Visual Basic* нема класичне библиотеке, већ постоје модули који се састоје од класа које имају методе које извршавају одређене функције.

Класа која нуди методе за израчунавање често коришћених математичких функција зове се *Math*. Класа има десетине метода, од којих су издвојене неке од чешће коришћених.



Назив функције	Опис
Abs	Враћа апсолутну вредност броја
Ceiling	Враћа већу заокружену целобројну вредност децималног или реалног броја. (7.4 ~ 8)
Cos	Враћа вредност косинуса угла задатог у радијанима.
Exp	Враћа вредност функције e^x за степен наведен у аргументу функције.
Floor	Враћа мању заокружену целобројну вредност децималног или реалног броја. (7.6 ~ 7)
Log	Враћа вредност логаритма са основом e за дати број.
Log10	Враћа вредност логаритма са основом 10 за дати број.
Max	Враћа већи од два броја.
Min	Враћа мањи од два броја.
Pow	Враћа вредност броја подигнутог за одређени степен.
Round	Враћа децималну вредност децималног или реалног броја са задатим бројем децимала.
Sign	Враћа целобројну вредност: 1, ако је аргумент број позитиван. 0, ако је аргумент функције 0 -1 ако аргумент функције негативан број
Sin	Враћа вредност синуса угла задатог у радијанима.
Sqrt	Враћа вредност квадратног корена броја.
Tan	Враћа вредност тангенса угла задатог у радијанима.



Задатак: Написати програм који за вредност степена коју уноси корисник исписује синус, косинус, тангенс и котангенс угла. Водити рачуна о конверзији из степена у радијане.

Формула за конверзију степена у радијане је

Math библиотека има дефинисану константу *Math.PI* у којој је смешта приближна вредност π . Такође, вредност константе e се може користити у програму навођењем *Math.E*.

Задатак је прилично једноставан - од корисника се тражи да унесе број у степенима, а затим коришћењем *Math* библиотеке исписује тражене вредности. Следи решење задатка.

1. `Sub Main()`
2. `Dim ugao As Double`
3. `Dim radijani As Double`
- 4.
5. `Console.WriteLine("Unesite ugao (u stepenima): ")`



```
6.      ugao = Val(Console.ReadLine())
7.      radijani = ugao * Math.PI / 180
8.
9.      Console.WriteLine($"Sinus: {Math.Sin(radijani)}")
10.     Console.WriteLine($"Kosinus: {Math.Cos(radijani)}")
11.     Console.WriteLine($"Tangens: {Math.Tan(radijani)}")
12.     Console.WriteLine($"Kotangens: {1 / Math.Tan(radijani)}")
13. End Sub
```

У програму су коришћене две променљиве, променљива *ugao* у коју се уписује вредност коју је унео корисник и *radijani* у коју се смешта израчуната вредност угла у радијанима. Све коришћене методе за извршавање тригонометријских функција морају да испред себе да имају префикс *Math* како би тачно било одређено којој класи припадају. Како у *Math* нема методе за котангенс, примењена је формула $\tan(x) = 1 / \text{ctg}(x)$ за добијање резултата.



БЕЛЕШКЕ



Наредбе гранања

У претходним поглављима биле су коришћене наредбе доделе, као и позиви метода неких елемената графичког корисничког интерфејса. Ове наредбе су се извршавале по редоследу у ком су писане и то свака наредба се извршавала тачно једном. Некада се извршавање неког блока наредби разликује у зависности од унетих података или неких других захтева задатка. Наредбе које омогућују ову функционалност називају се наредбе гранања или условне наредбе. Visual Basic има две врсте наредби гранања – `if/else` и `select case`.

If/else наредба

If наредба се извршава тако што се утврђује да ли је услов који следи после ове кључне речи тачан, ако јесте, онда се извршава наредба или блок наредби који извршава жељену акцију. Вредност услова, односно резултат израза у услову мора бити логичког типа (тачно или нетачно). У Visual Basic-у исход логичких оператора представљен је логичким константама **True** уколико је услов тачан, односно са **False** уколико услов није тачан. Тип података који може да складишти ове вредности је *boolean* тип података. У табели 1 дат је преглед релацијских оператора који се могу користити приликом писања услова у `if/else` наредби.

Оператор	Опис
=	Једнако
>	Веће
<	Мање
>=	Веће или једнако
<=	Мање или једнако
<>	Различито

Табела 1. Списак телационих оператора

Основни облик **if** наредбе је наредба која извршава блок наредби које се налазе између кључних речи **Then** и **End If** само у случају да када је услов тачан:

If *uslov* **Then**

 ' kod koji se izvršava kada je uslov tačan

End If



Задатак: Написати програм који за унуту целобројну вредност проверава да ли је унети број позитиван и ако јесте онда у конзоли приказује поруку да је корисник унео позитиван број.

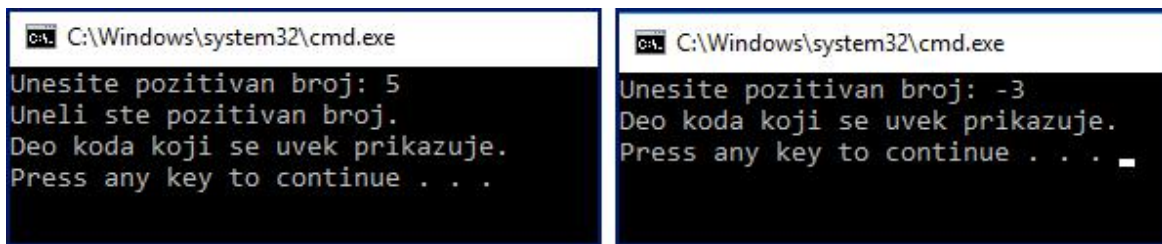
Решење задатка:.

1. **Sub** Main()



```
2.      Dim a As Integer
3.      Console.WriteLine("Unesite pozitivan broj: ")
4.      a = Val(Console.ReadLine())
5.      If a > 0 Then
6.          Console.WriteLine("Uneli ste pozitivan broj.")
7.      End If
8.      Console.WriteLine("Deo koda koji se uvek prikazuje.")
9.  End Sub
```

На линији 5 проверава се да ли је број већи од нуле. Уколико јесте онда се извршава низ наредби које се налазе између кључних речи **Then** и **End If**. Тај низ наредби се назива **блок**. На слици 1 дат је исход извршавања програма када корисник унесе позитиван и негативан број.



Слика 1. Пример извршавања задатка за унос позитивне и негативне вредности

На левом делу слике 1, приказан је случај када је унета вредност -3 порука, тада се порука „Uneli ste pozitivan broj“ не приказује. Наредба гранања која дефинише само акције које се извршавају када је услов тачан је проста наредба гранања.

Следећи облик if/else наредбе је:

```
If uslov Then
    ' kod koji se izvršava kada je uslov tačan
Else
    ' kod koji se izvršava kada uslov nije tačan
End If
```



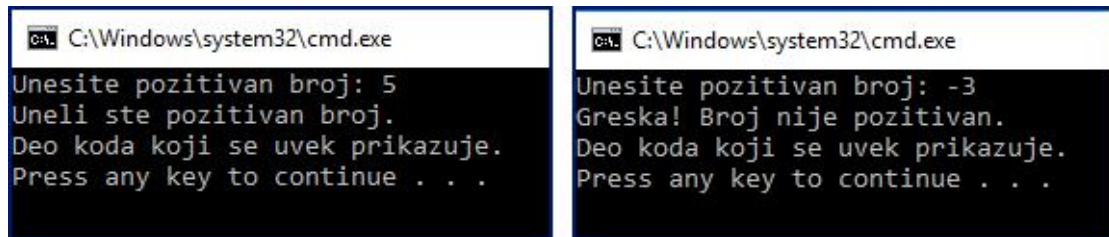
Задатак: Написати програм који за унуту целобројну вредност у конзоли испишује поруку поруку о томе да ли је унети број позитиван или не.

Решење задатка:

```
1.  Sub Main()
2.      Dim a As Integer
3.      Console.WriteLine("Unesite pozitivan broj: ")
4.      a = Val(Console.ReadLine())
5.      If a > 0 Then
```



```
6.         Console.WriteLine("Uneli ste pozitivan broj.")
7.     Else
8.         Console.WriteLine("Greška! Broj nije pozitivan.")
9.     End If
10.    Console.WriteLine("Deo koda koji se uvek prikazuje.")
11. End Sub
```



Слика 2. Пример извршавања задатка за унос позитивне и негативне вредности са проширеном наредбом

Када се апликација покрене биће исписана тачно једна порука. Уколико је услов тачан извршава се блок наредби који се налази између резервисаних речи **Then** и **Else**, а уколико није тачан онда ће се извршити низ наредби од **Else** до **End If**. Изглед екрана по извршењу задатка дат је на слици 2.

Неки проблеми захтевају провере више услова од којих даљи ток извршавања програма зависи. Једну класу ових проблема решава трећи облик `if/else` наредбе. Код ове наредбе постоји више услова, од којих први тачан услов ће условити извршавање низа наредби који припадају одговарајућем блоку тог дела `if` наредбе. Свака могућа реализација за назива се **грана**. Синтакса ове наредбе је:

```
If uslov1 Then
    ' blok naredbi koji se izvršava kada je uslov1 tačan
ElseIf uslov2 Then
    ' blok naredbi koji se izvršava kada je uslov2 tačan
.
.
.
ElseIf uslovN Then
    ' blok naredbi koji se izvršava kada je uslovN tačan
Else
    ' blok naredbi koji se izvršava kada nijedan od uslova nije
    tačan
End If
```

Број услова може бити произвољан. У сложеној условној наредби **Else** грана не мора да постоји.



Задатак: Написати програм који од корисника тражи да унесе целобројну вредност. У зависности од уноса програм треба да испише да ли је број позитиван, негативан или је корисник унео нулу.

Постоје три случаја – први је да је корисник унео позитиван број, други је да је унео негативан а трећи је да је унео нулу. Овај задатак биће решен на два начина – коришћењем **Else** гране, и без коришћења **Else** гране, само **Elseif** гране.

```
1. Sub Main()  
2.     Dim a As Integer  
3.     Console.WriteLine("Unesite pozitivan broj: ")  
4.     a = Val(Console.ReadLine())  
5.     If a > 0 Then  
6.         Console.WriteLine("Uneli ste pozitivan broj.")  
7.     ElseIf a < 0 Then  
8.         Console.WriteLine("Uneli ste negativan broj.")  
9.     Else  
10.        Console.WriteLine("Uneli ste nulu.")  
11.    End If  
12.    Console.WriteLine("Deo koda koji se uvek prikazuje.")  
13. End Sub
```

На линији 9 почиње **Else** грана која се извршава када корисник унесе нулу. Променљива „a“ као нумеричка целобројна вредност може имати само три стања, може бити – позитивна, негативна и нула. Тако да уколико корисник унесе број, то су три исхода која се очекују па није потребно писати трећи услов провере, али је и то могуће:

```
ElseIf a = 0 Then
```

Тернарна if наредба

Постоји специјални облик **if** наредбе који се зове тернарни облик. Овај облик се користи када нека променљива или израз треба да добије вредност у зависности да ли је услов тачан или није. Облик наредбе је:

```
izraz = If(uslov, vrednost za tačan uslov, vrednost za netačan uslov)
```



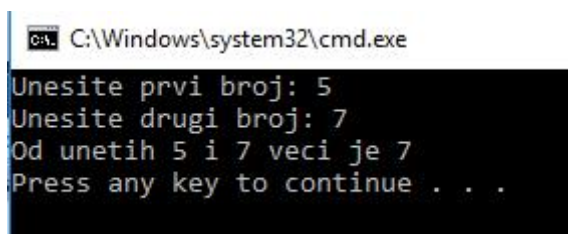
Задатак: Написати програм који за две унете целобројне вредности испишује већу од њих.

Овај задатак је урађен применом тенарног оператора. Излаз у конзоли исписаће већи од унетих бројева. Тенарни if користиће се директно наредби за испис.



```
14. Sub Main()  
15.     Dim a, b As Double  
16.     Console.Write("Unesite prvi broj: ")  
17.     a = Val(Console.ReadLine())  
18.     Console.Write("Unesite drugi broj: ")  
19.     b = Val(Console.ReadLine())  
20.     Console.WriteLine($"Od unetih {a} i {b} veći je { If(a > b, a,  
        b) } ")  
21. End Sub
```

На слици 3 приказан је резултат покретања програма за унете бројеве 5 и 7. Недостатак тенарног оператора је да узима сигурно једну од две вредности. У неким ситуацијама је ово непожељно и тада се мора користити if/else наредба која је претходно објашњена. Читаоцу се оставља за вежбу да напише програм који користи са одговарајућу if/else наредбу и у случају да су бројеви једнаки не врши доделу вредности, већ исписује поруку да су бројеви једнаки.



```
C:\Windows\system32\cmd.exe  
Unesite prvi broj: 5  
Unesite drugi broj: 7  
Od unetih 5 i 7 veci je 7  
Press any key to continue . . .
```

Слика 3. Извршавање задатка применом за тенарне if наредбе

Логички оператори

У претходним примерима коришћени су једноставни услови, међутим неки примери захтевају сложене услове, који се састоје из више простих услова. На пример за пријаву корисника на неки систем се очекује да корисник тачно унесе корисничко име и лозинку након чега се пријављује на систем. За решење овог задатка потребно је проверити да ли корисничко име одговара имену у некој бази корисника, као и да је унета лозинка тачна за датог корисника. За сложене услове се користе логички оператори. У табели 2 је дат преглед логичких оператора Visual Basic-a.

Оператор	Опис
And	Логички оператор, такође се користи код битских операција. Ако су оба услова која се везују тачна, цео израз је тачан. Оператор проверава тачност оба услова.
Or	Логички оператор, такође се користи код битских операција. Ако је било који од два услова која се везују тачан, онда је израз тачан. Оператор проверава тачност оба услова.
Not	Логички оператор, такође се користи код битских операција. Враћа негацију стања услова. Уколико је услов тачан, исход услова је нетачан и обрнуто.
Xor	Логички оператор, такође се користи код битских операција. Зове се ексклузивни Or оператор. Уколико су оба услова тачна или оба услова



	нетачна вредност израза је нетачна, у супротном вредност израза је тачна. Оператор проверава тачност оба услова.
AndAlso	Експлицитни логички And оператор. Не може да се примењује на битским операцијама.
OrElse	Експлицитни логички Or оператор. Не може да се примењује на битским операцијама.
IsFalse	Проверава да ли је логички израз False.
IsTrue	Проверава да ли је логички израз True.

Табела 2. Логички оператори у Visual Basic-у

Неки од облика за ове операторе дати су у наставку:

```
If uslov1 And uslov2 Then
If uslov1 Or uslov2 Then
If uslov1 Xor uslov2 Then
If uslov1 AndAlso uslov2 Then
If uslov1 OrElse uslov2 Then
If Not uslov1 Then
If IsFalse uslov1 Then
If IsTrue uslov1 Then
```

У if/else наредби може да буде произвољан број услова и могу се комбиновати оператори како би критеријум био задовољен.

```
If uslov1 And Not(uslov2 Or uslov3) Then
```

За груписање услова користе се заграде, јер оне имају највећи приоритет.



Задатак: Написати програм који од корисника тражи да унесе колико има година. Уколико корисник припада старосној групи од 13 до 18 година треба да се испише да је тинејџер, уколико има између 19 и 26 да се испише да је студент, док уколико има више од 26 година треба исписати да је радно способан.

У тексту задатка дати су опсези неких вредности. Када се тражи да нека вредност буде у неком опсегу, мисли се да вредност треба да буде већа од минимума а мања од максималне вредности (и може да укључује те вредности осим ако није експлицитно наведено да не може). У задатку се помињу три опсега, тако да се ради о сложеној условној наредби. Веза за сваку грану биће два услова – први који испитује да ли је број већи или једнак минималној вредности, и други где се испитује да ли је број мањи или једнак максималној вредности. Логички оператор који се користи за везивање може бити And или AndAlso.

```
1. Sub Main()
2. Dim godina As Integer
3. Console.WriteLine("Koliko imate godina?")
4. godina = Val(Console.ReadLine())
```



```
5.  
6.      If godina >= 13 And godina <= 18 Then  
7.          Console.WriteLine("Vi ste tinejdžer")  
8.      ElseIf godina >= 19 And godina <= 26 Then  
9.          Console.WriteLine("Vi ste student")  
10.     ElseIf godina > 26 Then  
11.         Console.WriteLine("Spremni ste za posao")  
12.     End If  
13. End Sub
```

Select Case наредба

Синтакса **Select Case** наредбе је:

```
Select Case izraz  
    Case vrednost1  
        ' kod koji se izvršava ukoliko je izraz jednak vrednost1  
    Case vrednost2  
        ' kod koji se izvršava ukoliko je izraz jednak vrednost2  
    .  
    .  
    .  
    Case vrednostN  
        ' kod koji se izvršava ukoliko je izraz jednak vrednostN  
    Case Else  
        ' kod koji se izvršava kada izraz nije jednak nijednoj od  
vrednosti  
End Select
```

Select Case наредба се извршава тако што се прво израчуна вредност израза, затим се израчуната вредност упоређује редом са вредностима који се налазе у **Case**–овима. Извршава се низ наредби који следи после **Case**-а чија вредност је једнака вредности израза, а после извршења овог блока наредби извршава се прва наредба после кључне речи **End Select**, дакле наредне вредности **Case**-ова се не проверавају.

За разлику од **if/else** наредбе **Select Case** наредба се састоји од грана које уместо услова зависе од вредности израза у наредби. Вредност може бити цели број, карактер или текст.



Задатак: Написати програм који од корисника тражи да унесе годину у којој је уписао студије и на основу ње добија план студирања. За године 2012, 2013, 2014 наставни план је „NPP 2012“ а за 2015, 2016 и 2017 „NPP 2015“. У случају да корисник унесе годину која није понуђена, приказује му се порука да за дату годину нема додељеног плана и програма.

Решење задатка:

```
1. Sub Main()  
2.     Dim godina As Integer  
3.     Console.WriteLine("Unesite godinu upisa studija: ")  
4.     godina = Val(Console.ReadLine())  
5.     Select Case godina  
6.         Case 2012  
7.             Console.WriteLine("Studirate po NPP 2012")  
8.         Case 2013  
9.             Console.WriteLine("Studirate po NPP 2012")  
10.        Case 2014  
11.            Console.WriteLine("Studirate po NPP 2012")  
12.        Case 2015  
13.            Console.WriteLine("Studirate po NPP 2015")  
14.        Case 2016  
15.            Console.WriteLine("Studirate po NPP 2015")  
16.        Case 2017  
17.            Console.WriteLine("Studirate po NPP 2015")  
18.        Case Else  
19.            Console.WriteLine("Nema dodeljenog studijskog plana za  
    datu godinu")  
20.    End Select  
21. End Sub
```

У овом решењу постоје одређена понављања. Иако се ради о само једној наредби која ће да се изврши за неколико случајева ово решење је доста дугачко. Следи решење истог задатка у коме су обједињени су случајеви код којих је обрада иста коришћењем једне **Case**. Овде је у једном **Case** наведено више вредности одвојених зарезом.

```
1. Select Case godina  
2.     Case 2012, 2013, 2014  
3.         Console.WriteLine("Studirate po NPP 2012")  
4.     Case 2015, 2016, 2017  
5.         Console.WriteLine("Studirate po NPP 2015")  
6.     Case Else
```



```
7.         Console.WriteLine("Nema dodeljenog studijskog plana za  
           datu godinu")  
8.     End Select  
9.
```

Ако је у **Case** потребно ставити нумерички интервал користити се кључна реч **To** између горње и доње вредности у **case** вредности.



Задатак: Написати Case наредбу која у зависности од променљиве „bodovi“ која може имати вредност од 0 до 100, исписује, у зависности од броја бодова, оцену коју је добио студент.

```
1.     Select Case bodovi  
2.         Case 0 To 50  
3.             Console.WriteLine("Ocena 5")  
4.         Case 51 To 60  
5.             Console.WriteLine("Ocena 6")  
6.         Case 61 To 70  
7.             Console.WriteLine("Ocena 7")  
8.         Case 71 To 80  
9.             Console.WriteLine("Ocena 8")  
10.        Case 81 To 90  
11.            Console.WriteLine("Ocena 9")  
12.        Case 91 To 100  
13.            Console.WriteLine("Ocena 10")  
14.     End Select
```

RadioButton и CheckBox елементи

Када се креира графички кориснички интерфејс добра је пракса да се грешка при уносу података минимизује. Коришћење неких, већ уграђених елемената омогућава реализацију овог захтева. На пример када корисник бира одређене понуђене опције могу се користити **RadioButton** и **CheckBox** елементи. Уколико је избор обавезан користи се **RadioButton** елемент док уколико је избор опциони користи се **CheckBox** елемент. Својство које је имају оба елемента је **Checked**. Када је вредност овог својства **True** тада је елемент изабран, односно уколико је вредност **False** елемент није изабран.

Провера да ли је елементу може *checked* стање промењено се реализовати на један од два начина:

1. **RadioButton** и **CheckBox** елементи имају догађај **CheckChanged** који се позива када се промени *checked* својство на кликнутом елементу.
2. У догађају неког другог елемента (дугмета, лабеле) може се проверити која је вредност *checked* својства елемента.



Уколико се изабере први начин, користи се *CheckChanged* догађај који се позива сваки пут када се промени вредност *Checked* својства, или *Click* догађај када корисник кликне да изабере опцију односно да је промени уколико се ради о *CheckBox* елементу.



Задатак: Написати део програма који се бави регистрацијом ангажовања спољних професора и сарадника, део који бира тип наставног особља, као и могућност дефинисања додатних ставки у уговору – да ли је уговор са 50% ангажовања на студијама или је гостујући предавач. Уколико ниједна од додатних опција није задата ангажовање је 30%. Дизајн форме дат је у наставку.

Са леве стране форме дати су *CheckBox* елементи који се налазе у палети са алатима, док су са десне стране *RadioButton* елементи. Када се користи *RadioButton* елемент, један од елемената је препоручено да буде већ изабран. У примеру то ће бити вредност за професора. У *Properties* палети својству *Checked* поставити вредност *True*. За *CheckBox* елемент у овом примеру није потребно да се поставља *Checked* својство али некада се и то ради, када је потребно иза интерфејса да корисник уклони опције које не жели.

CheckBox елемент са лабелом „50% angazovanja“ треба да има (Name) својство постављено на „chk50PAngazovanja“, док други елемент треба да има (Name) својство постављено на „chkPoPozivu“. (Name) својства за „RadioButton“ елементе имаће вредности „rbProfesor“, „rbAsistent“ и „rbSaradnik“ респективно. Дугме „Snimi podatke“ имаће име „btnSnimi“. За потребе решења задатка, уместо снимања података подаци ће бити приказани у искачућем прозору. За приказ података користи се метода „*MessageBox.Show*“ која приказује задати текст као искачући прозор. Порука ће се приказати када корисник кликне на дугме. У догађају за клик дугмета додати код који следи, а дугмету дати назив *prikaziPodatke*.

1. `Private Sub prikaziPodatke(sender As Object, e As EventArgs)
Handles btnSnimi.Click`
2. `Dim tekstZaPrikaz As String = ""`
3. `If rbProfesor.Checked = True Then`



```
4.      tekstZaPrikaz = tekstZaPrikaz & "Tip: Profesor" &
vbNewLine
5.      ElseIf rbAsistent.Checked = True Then
6.      tekstZaPrikaz = tekstZaPrikaz & "Tip: Asistent" &
vbNewLine
7.      Else
8.      tekstZaPrikaz = tekstZaPrikaz & "Tip: Saradnik" &
vbNewLine
9.      End If
10.
11.     If chk50PAngazovanja.Checked = True Then
12.     tekstZaPrikaz = tekstZaPrikaz & "50% angazovanja"
13.     ElseIf chkPoPozivu.Checked = True Then
14.     tekstZaPrikaz = tekstZaPrikaz & "Predavanja po pozivu"
15.     Else
16.     tekstZaPrikaz = tekstZaPrikaz & "30% angazovanja"
17.     End If
18.
19.     MessageBox.Show(tekstZaPrikaz)
20. End Sub
```

На линији 2 декларисана је променљива *tekstZaPrikaz* која на почетку има празан текст. На тај текст биће додаване вредности у зависности од тога која је опција изабрана. Од линије 3 до линије 9 је провера које од *RadioButton* елемената је изабрано. У пракси у зависности од изабране опције не узима се вредност из текста *RadioButton* већ се узима од неке структуре података, или у овом примеру додељује се текстуална вредност. *vbNewLine* је ознака за прелаз у нови ред.

У решењу задатка се испитује који је *CheckBox* изабран. Ако није чекирано, тј. потврђено ниједно поље, поставља се подразумевана вредност у променљиву *tekstZaPrikaz* (ово је реализовано у *Else* грани).



БЕЛЕШКЕ



Наредбе циклуса

У неким случајевима је за реализацију захтева програма потребно да се једна или више наредби понови више пута. Програмске петље омогућавају да се одређена група инструкција систематски понавља док се не испуни услов за прекид понављања. Услов за излазак из циклуса зове се и излазни критеријум. Излазни критеријум је најчешће: број извршених циклуса, достигнута тачност при рачунању у итеративном поступку, проналажење жељеног податка, крај датотеке и слично. Наредбе циклуса се могу поделити на основу места на ком се услов за излазак из петље налази на оне код којих је услов на почетку, у средини и на крају.

Прва наредба циклуса која ће бити разматрана има контролу услова на крају. То је **Do Loop** наредба. Синтакса ове наредбе је:

Do

Niz naredbi

Loop While *uslov*

или

Do

Niz naredbi

Loop Until *uslov*

Разлика између ове две наредбе је у употреби кључних речи *While* и *Until* оне се односе на то да ли ће се циклус понављати све док је испуњен услов (у случају да се користи *While*) или све док се не стекне неки услов (у случају да се користи *Until*). Треба водити рачуна како се поставља услов у петљи и која се од две могуће кључне речи користи. Контекст примене петље може сугерисати коју од ове две кључне речи треба изабрати. Низ наредби у телу петље ће се извршити бар једном, јер се услов за излазак из петље проверава на крају.

Ако је потребно да се прво испита услов, односно да се петља изврши и први и сваки наредни пут само кад је услов задовољен, користи се **While** наредба циклуса која услов испитује на почетку. Она има следећу синтаксу:

While *uslov*

Niz naredbi

End While

While петља је најопштија наредба циклуса и помоћу ње се могу реализовати све наредбе циклуса.

Веома често се користе наредбе циклуса које треба да се понове одређени број пута. На пример када се уноси одређени број података и када треба све те податке обрадити. Овај задатак се може реализовати помоћу претходно наведених наредби, али најчешће се користи наредба **for** која је за ове примене



веома погодна с обзиром да сама повећава или смањује бројач понављања циклуса. Синтакса ове наредбе је:

```
For бројач [As тип податка] = почетна вредност To крајња вредност  
    Низ наредби
```

Next

Заграде [] означавају да елементи који се у њима налазе нису обавезни. Тако на пример: *brojac* је променљива која може бити декларисана као локална променљива на пример: *For brojac As Integer = 1 To 10 ...* или ако је *brojac* променљива која је већ декларисана у претходном делу програма, наредба ће бити: *For brojac = 1 To 10* итд. У две претходно наведене наредбе се подразумевало да је корак 1 тј. да се бројач увећава после сваког извршавања циклуса за један. То не мора увек да буде тако. Коришћењем опционог дела *for* наредбе **Step** корак може да буде произвољан број.

Како се извршава *for* наредба? Прво се вредност бројача постави на почетну вредност и провери се да ли достигнута крајња вредност. Ако јесте, контрола програма прелази на прву наредбу после **Next**. Ако није, онда се изврши блок наредби *For* циклуса, увећа се бројач циклуса за величину корака (подразумевана вредност је 1), и опет се тестира да ли је дошло до краја циклуса, тј. да ли је вредност бројача већа од крајње вредности која је наведена у *for* наредби.



Пример 1: Написати програм који исписује бројеве од 1 до 10 коришћењем:

-  Do ... Loop While
-  Do ... Loop Until
-  For ... Next
-  While ... End

Пример за *Do Loop While*:

```
1. Sub Main()  
2.     Dim a As Integer = 1  
3.     Console.WriteLine("Do Loop While: Brojevi 1 do 10")  
4.     Do  
5.         Console.Write($"{a}    ")  
6.         a = a + 1  
7.     Loop While a <= 10  
8.     Console.WriteLine()  
9. End Sub
```



```
C:\WINDOWS\system32\cmd.exe
Do Loop While: Brojevi 1 do 10
1 2 3 4 5 6 7 8 9 10
Press any key to continue . . .
```

Слика 1. Извршавање примера

Пример за *Do Loop Until*:

```
1. Sub Main()
2.     Console.WriteLine("Do Loop Until: Brojevi 1 do 10")
3.     Dim a As Integer = 1
4.     Do
5.         Console.Write($"{a}  ")
6.         a = a + 1
7.     Loop Until a > 10
8.     Console.WriteLine()
9. End Sub
```

```
C:\WINDOWS\system32\cmd.exe
Do Loop Until: Brojevi 1 do 10
1 2 3 4 5 6 7 8 9 10
Press any key to continue . . .
```

Слика 2. Извршавање примера

Пример за *For Next*:

```
1. Sub Main()
2.     Console.WriteLine("For Next: Brojevi 1 do 10")
3.     For i As Integer = 1 To 10 Step 1
4.         Console.Write($"{i}  ")
5.     Next
6.     Console.WriteLine()
7. End Sub
```

```
C:\WINDOWS\system32\cmd.exe
For Next: Brojevi 1 do 10
1 2 3 4 5 6 7 8 9 10
Press any key to continue . . .
```

Слика 3. Извршавање примера

Пример за *While End*:

```
1. Sub Main()
2.     Console.WriteLine("While Next: Brojevi 1 do 10")
3.     Dim a As Integer = 1
```



```
4.      While a <= 10
5.          Console.Write($"{a}  ")
6.          a = a + 1
7.      End While
8.      Console.WriteLine()
9.  End Sub
```

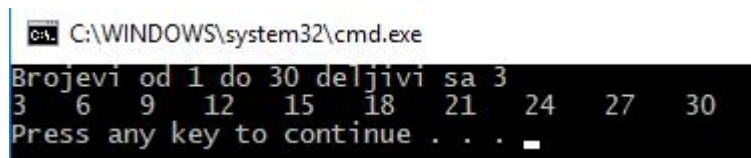
Наредбе Continue, Exit

У оквиру блока, тј. низа наредби које се извршавају у телу наредбе циклуса могу бити било које наредбе које су помињане: наредбе доделе, наредбе гранања и наредбе циклуса, као и неке наредбе које нису помињане. Прва наредба која ће бити разматрана је **Continue** наредба која прекида извршавање циклуса и контролу тока програма враћа на испитивање услова петље, тј извршава петљу за наредну итерацију. Она се може користити у свим претходно помињаним наредбама циклуса, с тим што се после continue наводи име петље: *Continue Do*, *Continue While*, *Continue For*.



Пример 2: Написати програм који за опсег од 1 до 30 исписује бројеве који су дељиви са 3 коришћењем While End петље.

```
1.  Sub Main()
2.      Dim broj As Integer = 1
3.      Console.WriteLine("Brojevi od 1 do 30 deljivi sa 3")
4.      While broj <= 30
5.          broj = broj + 1
6.          If Not broj Mod 3 = 0 Then
7.              Continue While
8.          End If
9.          Console.Write($"{broj}  ")
10.     End While
11.     Console.WriteLine()
12. End Sub
```



Слика 5. Извршавање примера

У примеру 2 дата је једна примена ове наредбе у оквиру *While* наредбе. У истом примеру се може видети и да *If* наредба може бити у телу петље. Као и да наредба у линији 9 неће бити извршена ако број није дељив са 3, јер ће онда наредба *Continue While*, проузроковати њено прескакање.



Једна од наредби која се може појавити у оквиру наредбе циклуса је и наредба **Exit**. Као и наредба *Continue* и она ће условити прекид извршавања наредби које следе иза ње у циклусу, али ће контролу извршавања програма пренети на прву наредбу после наредбе циклуса, а не на наредно извршавање петље. Другим речима наредба *Exit* врши терминирање, тј. окончавање циклуса. Користи се у ситуацијама кад је неки циљ постигнут или кад је даљи наставак рада циклуса немогућ. На пример у ситуацији када се тражи неки податак и он је нађен, или кад је добијена нула као делилац итд.

Као и код наредбе *Continue*, после навођења имена наредбе *Exit*, треба навести име петље на коју се она односи: *Exit While*, *Exit Do*, *Exit For*.



Пример 3: Написати програм који за симулира апликацију за пријаву корисника. Корисник има три покушаја да се пријави на систем. Уколико се корисник успешно пријави приказује му се порука о успешној пријави на систем. Уколико корисник не унесе добре податке, приказује му се грешка о лоше унетим подацима и нуди могућност да се поново пријави на систем. Уколико одустане од покушаја пријаве или прекорачи број покушаја исписује му се порука да није могуће да се пријави на систем.

Вредности за испитивање корисничког имена и лозинке су предефинисане и имају вредности „korisnik“ и „lozinka“ респективно.

```
1. Sub Main()  
2.     Dim brojPokusaja As Integer = 0  
3.     Dim prijavljen As Boolean = False  
4.     Dim odgovorPonovni As String  
5.     Dim korisnickoIme As String  
6.     Dim lozinka As String  
7.     Do  
8.         Console.Write("Unesite korisničko ime: ")  
9.         korisnickoIme = Console.ReadLine()  
10.        Console.Write("Unesite lozinku: ")  
11.        lozinka = Console.ReadLine()  
  
12.        If korisnickoIme.Equals("korisnik") And  
13.            lozinka.Equals("lozinka") Then  
14.            prijavljen = True  
15.            Exit Do  
16.        Else  
17.            brojPokusaja = brojPokusaja + 1  
18.            Console.WriteLine($"Podaci koje ste uneli nisu  
19.                validni. Broj pokušaja {brojPokusaja} od 3.")
```



```

18.         Console.WriteLine("Da li želite da pokušate ponovo? (Y/n)
        ")
19.         odgovorPonovni = Console.ReadLine()
20.         If Not (odgovorPonovni.Equals("Y") Or
odgovorPonovni.Equals("y")) Then
21.             Exit Do
22.         End If
23.     End If
24.     Loop While brojPokusaja < 3
25.
26.     If prijavljen = True Then
27.         Console.WriteLine("Uspešno ste se prijavili na sistem.")
28.     Else
29.         Console.WriteLine("Vaša prijava nije prihvaćena.")
30.     End If
31. End Sub
    
```

Слика 6. Пример извршавања програма - сви случајеви



Пример 4: Написати програм који израчунава вредност факторијала броја који задаје корисник све док корисник не унесе реч „крај“.

```

1. Sub Main()
2.     Dim n As Integer
3.     Dim unos As String
4.     Dim faktorijal As Integer
5.
6.     While True
7.         Console.WriteLine("Unesite broj (0 ili veći, reč 'kraj' za
        kraj): ")
8.         unos = Console.ReadLine()
9.
10.        If unos.Equals("kraj") Then
    
```



```
11.           Exit While
12.       End If

13.       faktorijal = 1
14.       n = Val(unos)
15.
16.       For i As Integer = n To 1 Step -1
17.           faktorijal = faktorijal * i
18.       Next
19.
20.       Console.WriteLine($"Faktorijal broja {n} je
    {faktorijal}")
21.   End While
22. End Sub
```

```
C:\Windows\system32\cmd.exe
Unesite broj (0 ili veci, rec 'kraj' za kraj): 5
Faktorijal broja 5 je 120
Unesite broj (0 ili veci, rec 'kraj' za kraj): 4
Faktorijal broja 4 je 24
Unesite broj (0 ili veci, rec 'kraj' za kraj): 0
Faktorijal broja 0 je 1
Unesite broj (0 ili veci, rec 'kraj' za kraj): 1
Faktorijal broja 1 je 1
Unesite broj (0 ili veci, rec 'kraj' za kraj): kraj
Press any key to continue . . .
```

Слика 7. Извршавање примера



БЕЛЕШКЕ



Низови

У претходном поглављу објашњено је на који начин се један блок наредби може поновити више пута применом наредби циклуса. Међутим, некад је потребно обавити исте радње, али над различитим променљивама. Једна од идеја би била да се ове променљиве именују истим именом са различитим бројем на крају имена као на пример: *vrednost1*, *vrednost2*,... а да им се приступа помоћи неког бројача *i*, па би на месту појављивања те променљиве у изразу писало *vrednosti*, међутим, ово није решење проблеме, јер би компајлер тражио променљиву која се зове *vrednost1*, а не би *i* схватио као бројач. Решење овог проблема је у увођењу сложене структуре података која се састоји из елемената истог типа, који су смештени у узастопним меморијским локацијама. Оваква структура се назива низ.

Статички низови

У Visual Basic-у низа се декларише као и све обична променљива, само што се после имена променљиве у загради наводи број који представља последњи индекс у низу:

Dim *imeNiza*(*brojelementa*) **As** **TipPodatka**.

Број елемената низа је опциони аргумент. Низови код којих се при декларацији наводи последњи индекс низа су **низови фиксне дужине или статички низови**.

У наставку су дати примери декларације низова:

```
Dim strData(21) As String
Dim numData(9) As Integer
Dim miscData(10) As Object
```

У првој линији кода декларисан је низ текстуалних података од **22 елемента**. Ово није случај код других програмских језика, већ је специфично за Visual Basic. Да бисмо разумели зашто је за дату декларацију дефинисан 21 елемент, треба да разумемо како се у меморији рачунара формира низ.

```
Dim strData(5) As String
```

У меморији рачунара формира (алоцира) се простор за низ на слици 1.



Слика 1. Пример меморијске алокације низа у рачунарској меморији

Када је низ декларисан онда му се могу доделити вредности. На располагању је 6 места на којима се подаци могу сместити. Сваки елемент низа има свој



индекс (или позицију). У већини актелних програмских језика први индекс у низу је 0.

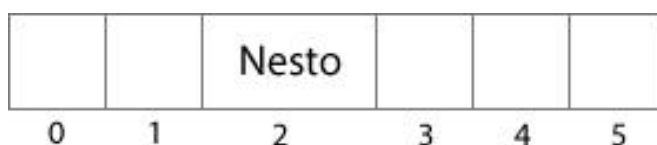
Рецимо да је потребно сместити податак у трећи елемент низа. Иницијализација елемента је слична као и иницијализација променљиве. Иницијализација се врши наредбом доделе

imeNiza(indeks) = vrednost.

У претходном примеру за strData додела вредности за трећи елемент је дата наредбом:

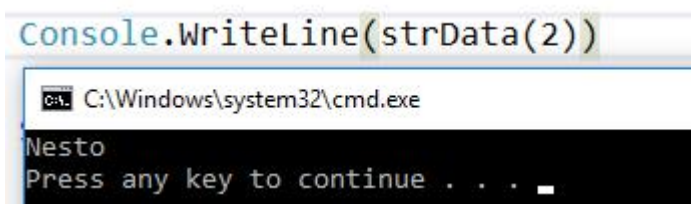
```
strData(2) = "Nesto"
```

У меморији рачунара сада је снимљена вредност (слика 2) и могуће јој је приступити коришћењем индекса „2“.



Слика 2. Пример смештене вредности „Nesto“ на индексу 2

Да би дохватили вредност низа потребно је навести име низа и у заградама индекс жељеног елемента (слика 3).



Слика 3. Приступ и испис вредности елемента са индексом 2 низа strData

Шта би се десило када би приступили елементу коме није додељена вредност? У зависности од типа елемената низа добила би се његова подразумевана вредност – за текст је празан стринг, за бројеве је 0, а за објекте је Nothing.

Када се наведе индекс низа који је већи од максималног броја индекса наведеног у декларацији јавља се грешка, јер меморијски простор коме би се приступало не припада низу. Рецимо да у strData желим да приступим елементу чији је индекс 10. У том случају јавиће нам се грешка (слика 4).



Слика 4. Приказ грешке након извођења наредбе Console.Write(strData(10))



IndexOutOfRangeException је грешка коју избацује програм и она се назива **грешком у току извршавања** (runtime) када није обрађена у оквиру програма, овај тип грешке доводи до прекида рада програма.

Постоји други начин дефинисања низова где се елементи одмах могу доделити низу. За овај тип низа није потребно дефинисати величину елемената. Декларација низа је следећа:

```
Dim imeNiza() As TipPodatka = { елементи низа раздвојени зарезом }
```

Пример једног низа са фиксним елементима дат је у наставку

```
Dim strData() As String = {"You", "Me", "Food", "Now"}
```

Иако су названи низовима фиксне дужине, могуће је да се њихове дужине прошире или смање и да се сачувају постојећи елементи. Принцип за проширење и скупљање низа исто је као и код низа динамичке величине.

Динамички низови

За разлику од статичких низова, **динамички низови** немају предефинисан број елемената. Овај тип низова се користи када је број елемената зависан од уноса корисника. Декларација динамичког низа је дата као **Dim imeNiza() As TipPodatka**. Низ декларисан на овај начин није употребљив односно није му могуће додати елементе.

Да би се динамички низ могао користити потребно је одредити број елемената. То се постиже коришћењем наредбе **ReDim** која датом низу поново додељује меморијски простор. Ова наредба се може користити и код статичких низова. У наставку следи креирање динамичког низа.

```
23. Dim dinNiz() As Integer
```

```
24. ReDim dinNiz(4)
```

```
25. dinNiz(1) = 5
```

```
26. Console.WriteLine(dinNiz(1))
```

После навођења **ReDim** наредба низ добија простор за смештање пет елемената. Шта се дешава ако се низу поново промени број елемената?

```
27. Dim dinNiz() As Integer
```

```
28. ReDim dinNiz(4)
```

```
29. dinNiz(1) = 5
```

```
30. Console.WriteLine("Pre: " & dinNiz(1))
```

```
31. ReDim dinNiz(6)
```

```
32. Console.WriteLine("Posle: " & dinNiz(1))
```

На слици 5 дат је излаз на конзоли.



```
C:\Windows\system32\cmd.exe
Pre: 5
Posle: 0
Press any key to continue . . .
```

Слика 5. Исход извршавања примера за додатни ReDim

Шта се догодило? ReDim наредба има и споредни ефекат. Када се низ прошири, бришу му и све вредности постојећих елемената. Често је потребно променити димензију низа, али и сачувати елементе у низу и после његовог проширења. Да би се ово постигло користе се наредбе ReDim и Preserve где је Preserve индикатор даје потребно сачувати претходни садржај. У претходном примеру треба изменити линију 5:

5. `ReDim Preserve dinNiz(6)`

Када се покрене апликација добија се излаз на слици на слици 6. А елементи низа су сачувани на новој меморијској локацији.

```
C:\Windows\system32\cmd.exe
Pre: 5
Posle: 5
Press any key to continue . . .
```

Слика 6. Исход извршавања примера са наредбом ReDim Preserve

Пролазак кроз низове коришћењем петљи

For петља

Када се низ састоји од већег броја елемената, за испис свих елемената није исплативо приступити сваком елементу појединачно коришћењем индекса (niz(0), niz(1), ...) већ обично се ради итерација коришћењем петљи где коришћењем итератора (i) приступамо елементу. За десет елемената низа петља ће се кретати у опсегу од 0 до 9 а приступ елементу низа биће у формату niz(i).

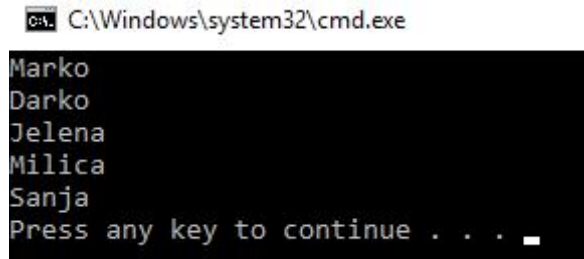
Узмимо пример листе имена студената. Имамо низ који има облик:

```
Dim imenaStudenata() As String = { "Marko", "Darko", "Jelena",  
"Milica", "Sanja" }
```

и желели би да испишемо та имена. Користићемо For петљу која ће имати почетну вредност 0 и крајњу вредност 4. Код за испис имена студената би изгледао овако:

```
6. For i As Integer = 0 To 4
7.     Console.WriteLine(imenaStudenata(i))
8. Next
```

Као излаз добићемо испис на слици 7.



Слика 7. Испис имена студената коришћењем For петље

И ово за сада функционише. Знамо да има 5 елемената, да индекси су од 0 до 4 па смо могли да поставимо крајњу вредност петље. Али у реалним примерима нама неће бити познато колико елемената корисник уноси. Треба нам неко својство или функција која ће нам рећи дужину датог низа. Својство које низови поседују зове се **Length** и вратиће нам број елемената у низу. Тако да код изнад се може изменити.

```
9. For i As Integer = 0 To imenaStudenata.Length - 1
10.     Console.WriteLine(imenaStudenata(i))
11. Next
```

Вероватно се питате зашто смо у петљи ставили `imenaStudenata.Length - 1` а не само `imenaStudenata.Length`. Својство `imenaStudenata.Length` вратиће нам вредност 5, а нама су индекси од 0 до 4. Да смо оставили само `imenaStudenata.Length` добили бисмо `IndexOutOfRangeException` зато што не постоји елемент на индексу 5.

For Each петља

Једна од петљи које нисмо помињали приликом рада са петљама је For Each петља. Разлог је што овај тип петље искључиво везан за итеративне типове података (низови, матрице, структуре ...). Петља има облик: **For Each ime iteratora As Tip podatka In ime niza.**

У For Each петљи постоји елемент који се зове **итератор**. Задатак тог елемента је да у текућој итерацији смести једну вредност из низа у себе. Најприближнији сценарио би била позиција мерења воћа и поврћа у продавници. Једна особа може на једном мерилу да мери воће и поврће. Остале особе чекају у реду (низ) да та особа заврши са мерењем а затим долази следећа особа.

Док код петље где нам је итератор индекс низа, For Each петља не води рачуна који је индекс елемента коме се приступа, већ који је елемент на реду. Због тога се ни не води рачуна о крају елемента, већ петља се завршава када нема следећег елемента.

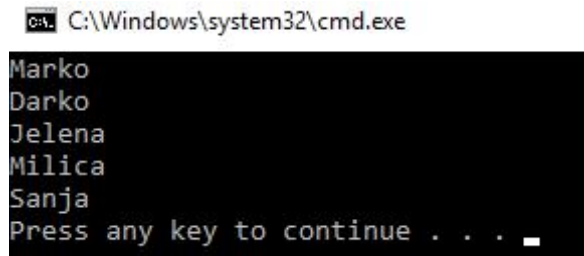
Пролазак са итератором који узима индексе даје већу флексибилност у виду проласка кроз низ (нпр. испис сваког другог елемента, испис низа уназад...) док итератор који узима елементе из низа мора да иде редоследом низа од почетка до краја.



У наставку дат је код за испис имена студената коришћењем For Each петље:

```
12. For ime As String In imenaStudenata
13.     Console.WriteLine(ime)
14. Next
```

Као излаз добићемо испис на слици 8.



Слика 8. Испис имена студената коришћењем For Each петље

Можемо да видимо да разлике између исписа студената коришћењем For петље и For Each петље нема.



Задатак: Користећи For петљу исписати имена студената почев од последњег у низу до првог елемента. Да ли исто можете урадити коришћењем For Each петље?

Сортирање низова

Када се корисницима презентују подаци, било у форми листи или статистика или неког другог прикладног вида презентације података, води се рачуна да ти подаци су прегледни и разумљиви за корисника. Ако узмете ранг листе за упис факултета оне су сортиране по броју бодова. Ако узмете спискове студената, обично су сортирани по броју индекса. Подаци у сировој форми не морају бити уређени. Корисник који уноси податке може да посматра различите изворе података па сам поредак уноса није сређен. Због оваквих случајева, али и случајева да када и имамо сређене податке по једном критеријуму а желимо да их уредимо по другом, користимо сортирање низова. Постоје различити алгоритми за сортирање низова. Неки су лакши за разумевање и имплементацију, неки не али исход је увек исти. Када радите сортирање било којег низа података, исход сортирања је **растући или опадајући** низ.

Bubble sort

Bubble sort једноставан алгоритам за сортирање низова. За сортирање већих количина података не сматра се као ефикасно решење

Bubble sort алгоритам пореди сваки пар елемената у низу и мења им места ако нису у траженом поретку све док цео низ није сортиран. За сваки елемент у листи алгоритам пореди пар елемената.



Извршавање bubble сорт алгоритма илустровано је следећим корацима:

1. Пореди $a[0]$ и $a[1]$. Уколико је $a[0]$ веће од $a[1]$. Уколико је $a[1]$ мање од $a[0]$, елементи мењају места.
2. Прелази се на следећи елемент (који сада може да садржи елемент који је заменио места са другим елементом) и пореди се са $a[2]$. Уколико је $a[2]$ мање од $a[1]$, елементи мењају места. Ово треба одрадити за сваки пар до краја низа.
3. Корак 1 и 2 поновити n пута, где је n број елемената низа.



Задатак: Коришћењем bubble сорт алгоритма сортирајте низ бројева 7,3,1,4,2 по растућем поретку (1,2,3,4,7).

Визуелни начин сортирања низа коришћењем алгоритма дат је на слици 9. Потребно је имплементирати алгоритам кроз Visual Basic.

	a[1]	a[2]	a[3]	a[4]	a[5]
i=5	7	3	1	4	2
j=1	3	7	1	4	2
j=2	3	1	7	4	2
j=3	3	1	4	7	2
j=4	3	1	4	2	7
	a[1]	a[2]	a[3]	a[4]	a[5]
i=4	7	3	1	4	2
j=1	1	3	4	2	7
j=2	1	3	4	2	7
j=3	1	3	2	4	7
	a[1]	a[2]	a[3]	a[4]	a[5]
i=3	1	3	2	4	7
j=1	1	3	2	4	7
j=2	1	2	3	4	7
	a[1]	a[2]	a[3]	a[4]	a[5]
i=2	1	2	3	4	7
j=1	1	2	3	4	7
	a[1]	a[2]	a[3]	a[4]	a[5]
i=1	1	2	3	4	7

Слика 9. Визуелни приказ сортирања коришћењем Bubble сорт алгоритма

```

15. Dim a() As Integer = {7, 3, 1, 4, 2}
16.
17. For i = a.Length() - 1 To 1 Step -1
18.     For j = 0 To i - 1
19.         If a(j + 1) < a(j) Then

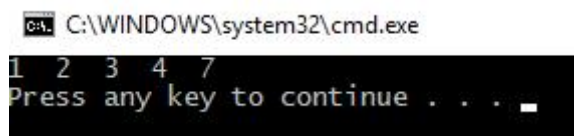
```



```
20.         Dim tmp As Integer = a(j)
21.         a(j) = a(j + 1)
22.         a(j + 1) = tmp
23.     End If
24. Next
25. Next
```

У линији 3 иде се од ***a.Length()* - 1** због два разлога: 1) У VB-у For петља ради над затвореним скупом, односно ако се напише нпр. 1 to 5 у опсег су укључене вредности и 1 и 5. Ако је дужина низа 5, иде се до вредности 4 јер елемент на индексу 5 не постоји и јавиће се грешка. До вредности 1 иде се због унутрашње петље, јер постоји услов ***i - 1*** за индексе елемента, па ако се стави да ***i*** бројач иде до 0, онда ***j*** бројач ће имати вредност -1 а то није могуће јер не постоје негативни индекси код низова.

У линијама 6,7,8 извршава се замена места елемената. Уводи се променљива tmp која чува стање првог елемента како се не би изгубила његова вредност када добије вредност следећег елемента. Резултат сортирања дат је на слици 10.



Слика 10. Резултат сортирања низа

Selection sort

Selection sort је алгоритам који узима позицију текућег елемента у низу а затим проверава да ли постоји елемент који је већи (или мањи од њега) и мења вредност позиције. Када се прође кроз цео низ, уколико је вредност позиције различита од позиције текућег елемента, елементу на пронађеној позицији и текућем елементу се мењају места.



Задатак: Коришћењем bubble sort алгоритма сортирајте низ бројева 7,3,1,4,2 по растућем поретку (1,2,3,4,7).

```
26. For i As Integer = 0 To a.Length - 2
27.     Dim pozicija As Integer = i
28.     For j = i + 1 To a.Length - 1
29.         If a(pozicija) > a(j) Then
30.             pozicija = j
31.         End If
32.     Next
33.     If pozicija <> i Then
34.         Dim tmp As Integer = a(i)
```



```
35.         a(i) = a(pozicija)
36.         a(pozicija) = tmp
37.     End If
38. Next
```

У линији 2 декларише и иницијализује променљива за позицију елемента која узима индекс текући индекс (i). На линији 3 пролази се кроз низ почев од наредног елемента и проверава се да ли је елемент на позицији већи од елемента на индексу j. Уколико јесте позиција мења вредност јер је нађен елемент који је мањи од текућег елемента. У линији 8 проверава се да ли се позиција променила од иницијално додељене вредности и уколико је то случај елементи мењају места.

Резултат сортирања дат је на слици 10.

Сортирање низа текстова

За разлику од бројева сортирање низа текста има другачије методе за испитивање да ли су два текста алфаветски сортирана. На нивоу овог курса бавићемо се латиничним текстовима, да бисмо избегли рад са стандардима за текст.



Задатак: Креирати форму као на видео демонстрацији *име видео фајла*.

Када корисник унесе име у поље текста, оно се додаје у низ имена. Свако унето име се додаје на крај текста другог поља за унос текста, а сортирани низ се приказује у трећем пољу у растућем поретку (A-Z).

У оквиру овог задатка биће демонстриране две ствари. Прва је како се проширује низ у зависности од уноса корисника, а друга ствар је како се по растућем поретку сортира низ текстова. За почетак потребно је дефинисати низ као глобалну променљиву и једну променљиву која броји колико је имена корисник унео у низ.

```
39. Public Class Form1
40.     Dim nizImena() As String
41.     Dim brojImena As Integer = 0
42. End Class
```

Следећи корак је дати одговарајућа имена графичким елементима. Поље за унос имена имаће (Name) вредност txtIme, поље за приказ несортираних имена txtImenaPrikaz, поље за приказ сортираних имена txtImenaSortirano, а дугме за додавање btnDodaj. Креирајте клик догађај дугмета и назовите га „dodajIme“. Унесите следећи код:

```
43. Private Sub dodajIme(sender As Object, e As EventArgs) Handles
    btnDodaj.Click
44.     ReDim Preserve nizImena(brojImena)
45.     brojImena = brojImena + 1
```



46. End Sub

У линији 2 ради се чување претходног стања низа и његово проширење. Када корисник први пут кликне на дугме, низ имена је 0, односно формира се низ nizImena који има само један елемент. У линији 3 повећава се број имена тако да са следећим кликом доћи ће до проширења низа за једно име. Ова линија биће на самом крају кода, односно додаће се код испред ње за додавање елемента и сортирања низа.

Измените претходни код са новим кодом:

```
47. Private Sub dodajIme(sender As Object, e As EventArgs) Handles  
    btnDodaj.Click  
48.     ReDim Preserve nizImena(brojImena)  
49.     nizImena(brojImena) = txtIme.Text  
50.     txtImenaPrikaz.Text += txtIme.Text + vbCrLf  
51.     brojImena = brojImena + 1  
52. End Sub
```

У линији 3 додаје се име на последњи индекс, који је уједно и број имена у низу. Први пут brojImena има вредност 0, односно низ се ставља на нулту позицију, следећи пут на прву итд. Ако вас буни због чега се у ReDim користи brojImena ако је то задњи индекс не заборавите да *Dim niz(n)* креира низ од ***n+1*** елемената. У линији 4 име се додаје на крај текста поља за приказ имена.

Следећи корак је имплементација сортирања низа. За алгоритам биће коришћен Selection сорт. Након објашњења како се сортирају стрингови, студентима се препоручује да покушају да добију исти резултат коришћењем bubble sort алгоритма.

Измените постојећи код:

```
53. Private Sub dodajIme(sender As Object, e As EventArgs) Handles  
    btnDodaj.Click  
54.     ReDim Preserve nizImena(brojImena)  
55.     nizImena(brojImena) = txtIme.Text  
56.     txtImenaPrikaz.Text += txtIme.Text + vbCrLf  
57.  
58.     For i As Integer = 0 To nizImena.Length - 2  
59.         Dim pozicija As Integer = i  
60.         For j = i + 1 To nizImena.Length - 1  
61.             If StrComp(nizImena(pozicija), nizImena(j)) = 1 Then  
62.                 pozicija = j  
63.             End If  
64.         Next  
65.         If pozicija <> i Then  
66.             Dim tmp As String = nizImena(i)
```



```
67.         nizImena(i) = nizImena(pozicija)
68.         nizImena(pozicija) = tmp
69.     End If
70. Next
71.
72.     txtImenaSortirano.Text = ""
73.     For Each ime As String In nizImena
74.         txtImenaSortirano.Text += ime + vbCrLf
75.     Next
76.
77.     brojImena = brojImena + 1
78. End Sub
```

Од линије 6 до линије 18 имплементира се selection sort алгоритам, цео код је искориштен из претходног примера уз одређене измене. Уместо низа *a*, користи се *nizImena*. Променљива *tmp* је сада типа String јер се сортира низ стрингова. Најважнија измена је на линији 9. За испитивање да ли је стринг после другог стринга, користи се функција **StrComp** која пореди два стринга и враћа једну од три вредности: -1 уколико је први стринг пре другог стринга, 0 уколико су стрингови исти и 1 уколико је стринг алфабетски после другог стринга. Ова функција пореди карактере по ASCII вредности. За имена *Marko* и *Marta* по ASCII провери М и М су исти као и а и г, када се пореде четврти карактери алфабетски као и по ASCII табели *k* је пре *t*, тако да ће вредност функције *StrComp* вратити -1.

У линији 20 ради се брисање постојећег садржаја поља за унос текста пошто се низ сваки пут исписује јер са сваким додатим именом потребно је поново сортирати низ. Студентима се саветује да покушају да оптимизују задатак тако да се низ не сортира сваки пут већ да се елемент уметне на одговарајуће место у низу.

У линијама 21 до 23 ради се испис имена у текстуално поље где се приказује сортирани низ. Пример извршавања задатка дат је на слици 11.



Form1

Goran

Dodaj

Marko
Ana
Jelena
Marta
Goran

Ana
Goran
Jelena
Marko
Marta

Слика 11. Пример извршавања програма за пет унетих имена



БЕЛЕШКЕ



Процедуре и модули

Модули, процедуре или потпрограми, представљају логичну функционалну целину, која се може засебно кодирати, превести и тестирати, али се не може засебно извршити, већ се извршава као део главног програма или другог потпрограма. Пренос управљања-извршавања наредби из једног модула-потпрограма у други назива се позив потпрограма. Потпрограм се може позивати из разних тачака једног програма или из разних програма. На крају сваког потпрограма налази се инструкција за повратак у програм којом се управљање враћа модулу из кога је потпрограм позван, и то увек на корак који следи иза позива потпрограма. Када се било која процедура изврши, програм се наставља. VB.net има два типа процедура:

- Функције - након извршавања наредби, постоји неки резултат који функција враћа
- Процедуре - Функције које не враћају никакву повратну вредност

У програму процедуре позива се када постоји неки код који се више пута понавља или ако је код издвојен у смислену програмску целину. Процедура је само блок кода који се може позвати и све комплексне апликације састоје се од целина које користе процедуре. Процедуре су већ коришћене у претходним вежбама, један од њих је процедура Val(), или StrReverse(). Именовање функција као и именовање променљивих је важно да би некемо ко чита код могао да разуме шта он треба да ради. У наставку је пример кода једног потпрограма:

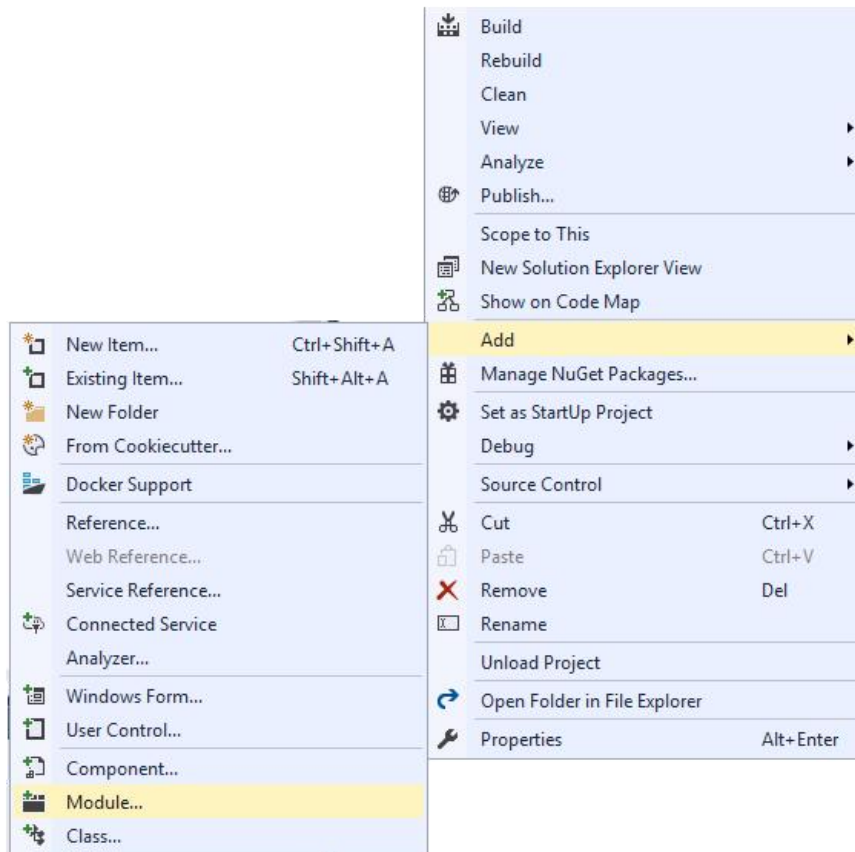
```
79. Dim t() As Double = { 300, 400, 250, 120, 110 }
80. Dim s As Double = 0
81. For i As Integer = 0 To t.Length - 1
82.     s += t(i) - t(i) * 0.3
83. Next
```

Да ли имате идеју шта овај код ради? Ово је лоше написан код. Код је потребно да буде довољно описан тако да програм који га је написао, као и особа која сутра дан га користи може да разуме чему дати код служи. Због овога се уводе конвенције именовања догађаја, променљивих и графичких елемената. Код претходног примера коригован је да приближније опише функционалност:

```
84. Dim nizBrutoCenaPosla() As Double = { 300, 400, 250, 120, 110 }
85. Dim netoZaradaZaIsplatu As Double = 0
86. Dim brojPoslova As Integer = nizBrutoCenaPosla.Length - 1
87. Dim KOEFICIJENT_POREZA As Double = 0.3
88. For i As Integer = 0 To brojPoslova
89.     netoZaradaZaIsplatu += nizBrutoCenaPosla(i) -
        nizBrutoCenaPosla(i) * KOEFICIJENT_POREZA
90. Next
```



Сада би код требао да је разумљивији неке ко чита овај код. Проблем са овим делом кода је што може да израчуна само једну задају за једног запосленог. Потпрограми имају карактеристику да могу да се позову произвољан број пута за произвољан број запослених. Због овога ови потпрограми се смештају у посебне програмске целине које су зову **модули**. Да би се додао модул у пројекат у *Solution Explorer*-у потребно је кликнути десним кликом миша на име пројекта, а затим у менију изабрати ставку *Add* и изабрати опцију *Module...* (слика 1). Уколико модул не представља смислену целину онда се он именује као *Procedure.vb*.



Слика 1. Додавање модула у пројекат

Када се креира нова конзолна апликација прва датотека која се са њом креира је *Module1.vb*. Модул може да садржи и процедуру *Main* и код конзолних апликација она је главна процедура из које се покреће програм. Само један модул у једном пројекту може да има *Main* процедуру.

Ако се нека процедура користи у другим модулима или класама, довољно је да се само једном напише и преведе у једном модулу и да је проглашена за јавну процедуру (*Public*), онда је она доступна свим датотекама у тој апликацији. Ако се један потпрограм, не само независно пише, већ и независно преводи, онда он може бити позван из било ког програма, и тада је он екстерни. Екстерни потпрограми могу формирати библиотеке. Када је једном написан и истестиран, екстерни потпрограм може се користити било када и у било ком програму, па се



тима знатно олакшава писање сложених програма. Без обзира на начин имплементације, процедуре се приликом позивања најчешће морају проследити и неки подаци неопходни за њихово извршавање. То су параметри, а они се могу пренети из позивајућег програма на два начина: преношењем копије податка (*ByVal* - по вредности) и преношењем референце на податке и објекте (*ByRef* - по референци). Сваки од ових начина има своје предности и недостатке.

Свака процедура има свој потпис. Потпис процедуре састоји се од: видљивости процедуре, типа процедуре (функција, процедура), имена процедуре, опционо од улазних аргумената и типа повратне вредности уколико је тип процедуре функција. Примери потписа процедуре дати су у наставку:

1. Процедура која је јавно видљива, има име *odstampaj* и нема улазне аргументе

```
Public Sub odstampaj()
```

2. Функција која је јавно видљива, има име *formatiraj*, нема улазне параметре, а повратна вредност је типа *String*.

```
Public Function formatiraj() As String
```

3. Процедура која је јавно видљива, има име *ispisiPoruku* има један улазни аргумент послат по вредности који има име *poruka*.

```
Public Sub ispisiPoruku(ByVal poruka As String)
```

4. Функција која је јавно видљива, има име *pomocnaFunkcija* која има два улазна аргумента са именима *txtPrvi* и *txtDrugi* и типа *TextBox* послатих по референци и има повратну вредност *Double*.

```
Public Function pomocnaFunkcija(ByRef txtPrvi As TextBox, ByRef  
txtDrugi As TextBox) As Double
```



Задатак: Креирати модул *Procedure.vb*. У датом модулу написати процедуру која за низ зарада запосленог израчунава нето вредност исплате.

У уводном делу поглавља дат је код који обрачунава нето исплату за низ вредности зарада. Сада ће тај код бити модификован да може да се извршава независно за било коју низ зарада.

Пре писања кода у модулу потребно је дати одговоре на следећа питања:

- Да ли пишемо функцију или процедуру?
- Да ли је функција/процедура коју пишемо јавно видљива?
- Који су аргументи функције/процедуре?
- Да ли се аргументи шаљу као копија вредности или референца?

Да ли пишемо функцију или процедуру? На основу задатка можемо закључити да за прослеђен низ зарада израчунавамо укупну нето вредност исплате. Када год се у захтеву клијента/спецификације очекује да се на основу



неких података добије нови податак, пише се функција. Уколико нема формирања нових података (испис података, сортирање итд.) пише се процедура. У овом примеру пишемо процедуру.

Да ли је функција/процедура коју пишемо јавно видљива? Кратак одговор је „да“. Уколико функција или процедура није помоћна, она је увек јавно видљива. Под помоћним функцијама/процедурама сматрају се оне које немају самосталну сврху већ служе као компонента неке веће функције/процедуре.

Који су аргументи функције/процедуре? Ово зависи од функционалности. Одговор на ово питање се решава одговором на следеће питање „Шта је од података потребно дати функцији/процедури да би она успешно обавила задатак?“. Ако погледамо решење примера писано императивним путем дате су следеће променљиве: *nizBrutoCenaPosla*, *netoZaradaZalsplatu*, *brojPoslova*, *KOEFICIJENT_POREZA* и *i*.

nizBrutoCenaPosla је аргумент јер је потребно знати који су послови обављани. *netoZaradaZalsplatu* је резултат обрачуна, што значи да није аргумент јер није унапред познат, већ је резултат обрачуна. *brojPoslova* може се проследити као аргумент, али ако погледате начин на који је вредност добијена, не морамо га проследити јер га увек можемо добити на основу дужине низа бруто зарада. *KOEFICIJENT_POREZA* нам је потребан за обрачун зарада и требамо га проследити као аргумент. *i* је помоћна променљива коду петље и не морамо је проследити као аргумент.

Када смо одредили аргументе које шаљемо остаје још да се да одговор на питање - **Да ли се аргументи шаљу као копија вредности или референца?**

Одговор на ово питање је једноставан. Уколико мењате директно нешто над вредности коју шаљете као аргумент – аргумент се шаље као копија вредности, ако да – аргумент се шаље по референци.

Остало је још дати назив функцији. Назив функције треба да опише функционалност функције на основу назива и аргумената. Име функцији у примеру биће *izracunajNetoZaradu*, а имена аргумената биће *nizBrutoZarada* и *koeficijentPoreza*. Крајњи облик потписа је:

```
Public Function izracunajNetoZaradu(ByVal nizBrutoZarada() As Double,
koeficijentPoreza As Double) As Double
```

На основу имена и аргумената можемо недвосмислено рећи да ова функција израчунава нето зараду за низ бруто зарада и неки коефицијент пореза. И током овог курса препорука је да се трудите да све функције/процедуре које именујете, именујете смислено. Код за процедуру дат је у наставку:

91.

```
Public Function izracunajNetoZaradu(ByVal nizBrutoZarada() As Double, koeficijentPoreza As Double) As Double
```
92.

```
Dim netoZaradaZaIsplatu As Double = 0
```
93.

```
Dim brojPoslova As Integer = nizBrutoZarada.Length - 1
```



```
94.     For i As Integer = 0 To brojPoslova
95.         netoZaradaZaIsplatu += nizBrutoZarada(i) -
           nizBrutoZarada(i) * koeficijentPoreza
96.     Next
97.     Return netoZaradaZaIsplatu
98. End Function
```

Да би се из функције вратила нека вредност/објекат користи се резервисана реч **Return**. Процедуре немају *Return* у свом коду као што се може видети са процедуром *Main*.

У главном програму потребно је позвати и тестирати креирану процедуру. Позиве функција и процедура смо до сада већ радили, али нисмо се детаљније коментарисали како се позивају. За позив било које функције/процедуре из модула који је део пројекта довољно је написати *имефункције(аргументи)* или *имефункције()* уколико аргумената нема. На слици 2 дат је процес писања кода за позив функције. Можете видети да Intellisense даје изглед потписа функције како би програмер знао шта да проследи од вредности.



Слика 2. Intellisense приказ потписа функције

Код за главни програм у ком је тестирана функција дат је у наставку:

```
99. Sub Main()
100.     Dim netoZarada As Double = izracunajNetoZaradu({300, 400,
        250, 120, 110}, 0.3)
101.     Console.WriteLine($"Neto zarada je {netoZarada}")
102. End Sub
```



Задатак: Креирати модул *Modifikacije.vb*. У датом модулу написати процедуру *promeniSvojstva* која за прослеђено дугме мења његову боју позадине, боју текста и величину и тип фонта. Боја позадине дата је као *RGB* комбинација 25, 118, 210, боја фонта као *RGB* комбинација 245, 245, 245. Фонт ће бити *Arial* величине 12em.

Процедуру тестирати над произвољним дугметом у оквиру догађаја *Form Load* главне форме.

У оквиру овог примера биће демонстрирана употреба *ByRef*, односно слања аргумената по референци. Како се у тексту задатка тражи да се промене својства дугмета које се прослеђује онда се дугме неће слати по вредности, односно његова копија јер се директно мењају његова својства.

Код за процедуру дат је у наставку:

```
103. Public Sub promeniSvojstva(ByRef dugme As Button)
104.     dugme.BackColor = Color.FromArgb(25, 118, 210)
```



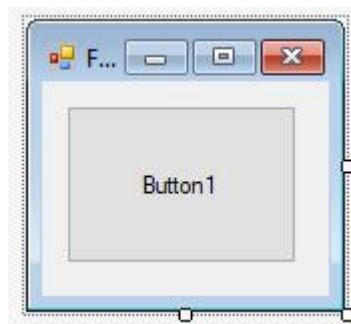
```
105.     dugme.ForeColor = Color.FromArgb(245, 245, 245)
106.     dugme.Font = New Font("Arial", 12)
107. End Sub
```

На линији 1 дугме се шаље као тип податка *Button*. До сада није било речи о типовима података графичких елемената. Да бисте знали који је тип податка графички елемент то можете проверити у *ToolBox* палети. Име које је дато уз елемент је и његов тип податка. Тако у овом примеру је за дугме које се прослеђује тип податка *Button* а користи се *ByRef* да се проследи по референци.

На линијама 2 и 3 позива се метода *FromArgb* из модула *Color* која на основу три аргумента које представљају јачину присуства црвене, зелене и плаве боје у опсегу од 0 до 255 респективно.

На линији 4 формира се вредност фонта на основу назива фонта и величине дате као *em* јединица (ширина латиничног слова *m*). Креира се на основу класе *Font*. О класама ће бити речи у наредном поглављу.

На слици 3 дат је дизајн форме на којој ће бити тестирана процедура. Име дугмета за тестирање је *btnZaPromenu*.

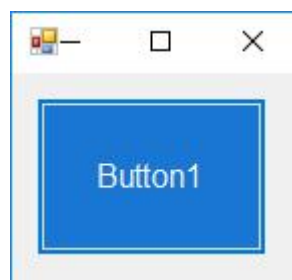


Слика 3. Дизајн демонстрационог примера

У догађају форме *Form Load* унесите следећи код:

```
108. Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
    MyBase.Load
109.     promeniSvojstva(btnZaPromenu)
110. End Sub
```

Након покретања програма требало би да добијете дизајн форме као на слици 4.



Слика 4. Дизајн демонстрационог примера након измена својстава



Проблем са овим начином писања модула је што су вредности непроменљиве. Бољи приступ је да се кориснику дозволи да проследи низ вредности које одговарају својствима које мења. У оквиру овог курса још нису рађене структуре података па ће бити коришћен низ фиксних вредности за демонстрацију овог примера.



Задатак: У модулу *Modifikacije.vb* изменити процедуру *promeniSvojstva* да поред дугмета прихвата и низ стрингова. На основу низа стрингова који се састоји од три елемента променити својства дугмету.

Први стринг биће вредности боја за боју позадине раздвојене зарезом. Други стринг ће имати исту структуру али за боју текста. Трећи стринг ће имати назив фонта и величину слова раздвојене зарезом.

Креирање својстава боје и фонта креирати коришћењем помоћних функција *bojaIzStringa* и *fontIzStringa* које је потребно написати.

Први корак је креирање помоћних функција у модулу пре измене главне методе. Ове функције су предвиђене да буду помоћне тако да ће њихова видљивост бити *Private*. Обе функције враћају вредности тако да неће бити процедуре. У наставку дат је код за помоћну процедуру која враћа боју:

```
111. Private Function bojaIzStringa(ByVal rgbString As String) As
    Color
112.     Dim izdvojeneJacine() As String = rgbString.Split(",")
113.     Dim r As Integer = CInt(izdvojeneJacine(0).Trim())
114.     Dim g As Integer = CInt(izdvojeneJacine(1).Trim())
115.     Dim b As Integer = CInt(izdvojeneJacine(2).Trim())
116.     Return Color.FromArgb(r, g, b)
117. End Function
```

Дата функција враћа вредност која је типа *Color*. Да би се добиле јачине боја потребно је стринг из низа поделити по зарезу, а затим те вредности претворити у целобројне вредности. Затим се формира боја са функцијом *FromArgb* и враћа назад као вредност типа *Color*. Метода за фонт је слична:

```
118. Private Function fontIzStringa(ByVal fontString As String) As
    Font
119.     Dim izdvojenaSvojstva() As String = fontString.Split(",")
120.     Dim imeFonta As String = izdvojenaSvojstva(0).Trim()
121.     Dim velicina As Integer = CInt(izdvojenaSvojstva(1).Trim())
122.     Return New Font(imeFonta, velicina)
123. End Function
```

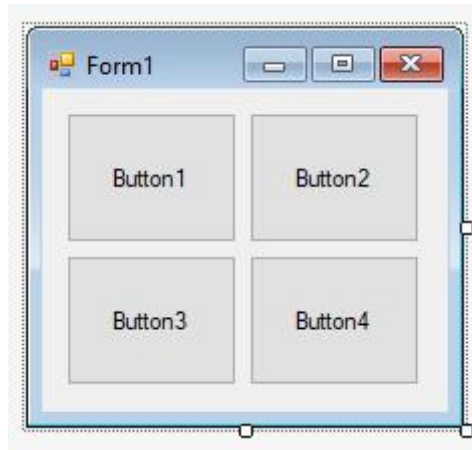
Наредни корак је измена процедуре *promeniSvojstva*:

```
124. Public Sub promeniSvojstva(ByRef dugme As Button, ByVal
    svojstva() As String)
```



```
125.     dugme.BackColor = bojaIzStringa(svojstva(0))
126.     dugme.ForeColor = bojaIzStringa(svojstva(1))
127.     dugme.Font = fontIzStringa(svojstva(2))
128. End Sub
```

За тестирање измењене процедуре измените постојећи дизајн. Обришите код из *Form Load* догађаја и на форми додајте уместо старог четири нова дугмета (слика 5).

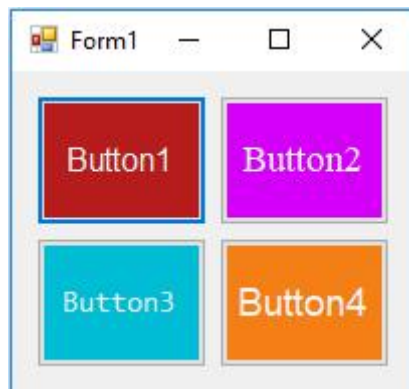


Слика 5. Дизајн демонстрационог примера

У *Form Load* догађају сада додајте следећи код:

```
129. Private Sub Form1_Load(sender As Object, e As EventArgs)
      Handles MyBase.Load
130.     promeniSvojstva(Button1, {"183, 28, 28", "245, 245, 245",
      "Arial, 12"})
131.     promeniSvojstva(Button2, {"213, 0, 249", "245, 245, 245",
      "Times New Roman, 14"})
132.     promeniSvojstva(Button3, {"0, 188, 212", "245, 245, 245",
      "Consolas, 11"})
133.     promeniSvojstva(Button4, {"245, 127, 23", "245, 245, 245",
      "Arial, 14"})
134. End Sub
```

Након покретања апликације требало би да добијете дизајн као на слици 6.





Слика 6. Дизајн демонстрационог примера након измена својстава



БЕЛЕШКЕ



Сложене структуре података

Објектно-оријентисано програмирање (ООП) представља новији приступ за решавање проблема на рачунару којим се решење тражи на природан начин који би се применио и у свакодневном животу. Нагласак у објектно-оријентисаном програмирању није на рачунарским поступцима него на објектима - софтверским елементима са подацима и методима који могу да делују једни на друге.

Објектно-оријентисано програмирање се своди на креирање различитих класа објеката којима се на природан начин моделира проблем који се решава. Притом, софтверски објекти у програму могу представљати не само реалне, него и апстрактне податке за одговарајући проблем.

Објектно-оријентисане технике, у принципу, могу се користити у сваком програмском језику. То је последица чињенице што објекте из стандардног угла гледања на ствари можемо сматрати обичном колекцијом променљивих и функција које манипулишу тим променљивим. Међутим, за ефективну реализацију објектно-оријентисаног начина решавања проблема је врло важно имати језик који то омогућава на непосредан и елегантан начин.

Класе и објекти

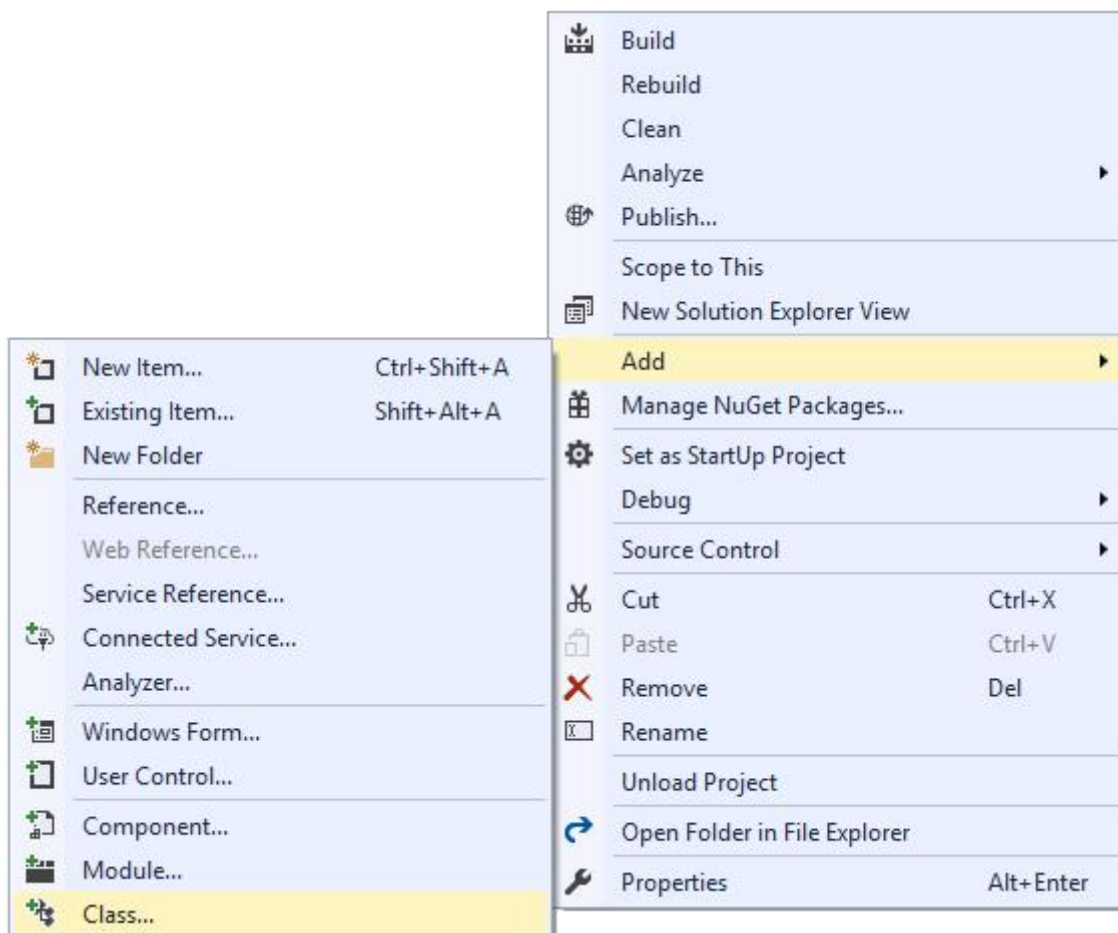
Класа је један од основних појмова у објектно оријентисаној парадигми. Представља логичку конструкцију којом се дефинише облик и понашање објекта. Било који концепт који треба да се реализује у неком објектно оријентисаном програмском језику мора да се енкапсулира у класу. Класа се може представити и као структура података којој су додате неке нове функционалности – методе али класа представља и нови, кориснички дефинисани тип. **Објекти** су чланови, тј. примерци класа који одговарају објектима из реалног света за разлику од саме класе која је апстракција. Објекти су повезани преко релација, а основна три типа ових релација су: **поткласе** (*subclasses*), **контејнери** (*containers*) и **сарадници**, колаборатори-кооперативне класе (*collaborators*).

Класа је програм који се не покреће самостално. Позива се креирањем променљиве истог типа као што је она. Затим се могу користити функционалности које она излаже позивањем чланова класе помоћу ове променљиве.

Како се класе најчешће дефинишу на основу објеката из стварног живота онда оне имају својства и понашања која су у фази пројектовања откривена код реалних објеката, а која су од значаја за остваривање циљева који се постављају пред апликацију. Наравно могу се додати и понашања која реални објекат нема а која доприносе ефикасности обраде (интелигентни објекти). Методи и својства класе, као и њени догађаји, чине интерфејс класе.



Класа се додаје у пројекат кликом десног дугмета миша на пројекат у *Solution Explorer*-у, затим избором менија *Add* и подменију избором опције *Class* (слика 1).



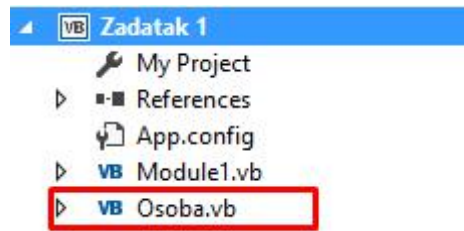
Слика 1. Додавање класе у пројекат

Конвенција код именовања класа је да треба њихово име треба да почне са великим словом и треба што приближније да опише објекат који ће бити креиран. Примери имена су: *Student*, *Interval*, *Oscena* итд.



Задатак: Креирати класу *Osoba* која има атрибуте за име, односно атрибуте за презиме особе. У конзолној апликацији креирати објекат класе, доделити вредност атрибутима и исписати информације за дати објекат.

За дати задатак потребно је додати класу *Osoba*. Име датотеке за класу биће исто као име класе са екстензијом *vb* која говори да се ради о датотеци која садржи Visual Basic код (слика 2).



Слика 2. Класа *Osoba* додата у пројекат

Класе се састоје од атрибута и метода. **Атрибути** су поља која чувају вредности за креирани објекат класе. За разлику од променљивих које током извршавања програма чувају једну вредност (која се може мењати), атрибут у класи представља део шаблона који објекат треба да испоштује. Атрибути се дефинишу као променљиве али за разлику од променљивих имају стања видљивости:

- Јавно (*Public*) стање - вредност овог поља ове видљивости може се мењати директним позивом.
- Приватно (*Private*) стање - вредност поља ове видљивости може се мењати само унутар метода његове класе.
- Заштићено (*Protected*) стање - вредност поља ове видљивости може се мењати унутар његове класе и класа које наслеђују ту класу.

Методе су функције које су груписане функције/рутине које могу да раде са атрибутима класе као и вредностима које се прослеђују као аргументи. Методи представљају имплементацију понашања објеката.

У траженом задатку атрибут *ће* за почетак бити јавно видљиви. Декларација атрибута дата је у формату: **Видљивост** **име атрибута** **As** **Тип податка**. Код за класу *Osoba* дат је у наставку:

```
135. Public Class Osoba
136.     Public ime As String
137.     Public prezime As String
138. End Class
```

Следећи корак је декларисање објекта класе. Декларација објекта је иста као и декларација било које променљиве, а за тип податка користи се име класе. Поред декларације објекат треба да се и иницијализује. Ово се разликује од примитивних типова података јер се за иницијализацију објекта мора користити резервисана реч **New**. Командом *New* је креиран нови објекат класе *Osoba* и она се користи да поред креирања објекта класе, најчешће, подеси почетне (подразумеване) параметре те класе. Тако, на пример, при креирању новог објекта типа дугме, он има сиву боју, тамну сенку на дну и са десне стране дугмета, светлу сенку са горње и леве стране дугмета, својство *Name* је *ButtonX*, где је *X* редни број до сада креираних дугмади итд. Сва та почетна подешавања се дефинишу унутар процедуре *New* унутар класе. Формат иницијализације објекта је **име променљиве** = **New** **Име класе**().



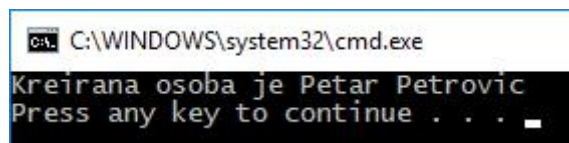
У главном програму потребно је имплементирати следећи код.

```
139. Sub Main()  
140.     Dim petar As Osoba  
141.     petar = New Osoba()  
142. End Sub
```

На овај начин формиран је објекат класе *Osoba*, променљива чији је назив *petar*. Као и код других променљивих њено име је произвољно и важи иста конвенција именовања, па је самим тим логично да име променљиве буде нешто што ће описати особу. Дати код само издваја меморијски простор за чување вредности променљиве али сама променљива нема вредности додељене својим атрибутима. Како је у класи наведено да су они јавно видљиви, онда им се може директно приступити. Директно приступање атрибутима је **објекат.атрибут**. У наставку је дат код за доделу вредности атрибутима и приступање њиховим вредностима.

```
143. Sub Main()  
144.     Dim petar As Osoba  
145.     petar = New Osoba()  
146.  
147.     petar.ime = "Petar"  
148.     petar.prezime = "Petrović"  
149.     Console.WriteLine($"Kreirana osoba je {petar.ime}  
        {petar.prezime}")  
150. End Sub
```

Покретање следећег програма даће испис дат на слици 3.



Слика 3. Пример извршавања програма



Задатак: Модификовати претходни програм тако да се испис имена и презимена реализује позивом методе.

Методe имају исте карактеристике као и функције/процедуре. Додатна функционалност је да се у методама може приступити атрибутима из класе. У наставку је дат код за проширену класу *Osoba*.

```
151. Public Class Osoba  
152.     Public ime As String  
153.     Public prezime As String  
154.  
155.     Public Function ispisi() As String
```

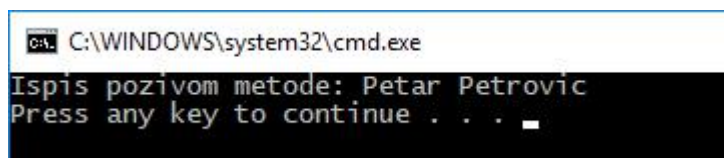


```
156.         Return $"{Me.ime} {Me.prezime}"
157.     End Function
158. End Class
```

За приступање атрибутима користи се резервисана реч **Me**. *Me* је реч која се односи на класу за коју се атрибут позива. Приступ јавној методи је исти као и приступ јавном атрибуту. Модификовани програм дат је у наставку:

```
159. Sub Main()
160.     Dim petar As Osoba
161.     petar = New Osoba()
162.
163.     petar.ime = "Petar"
164.     petar.prezime = "Petrović"
165.     Console.WriteLine($"Ispis pozivom metode: {petar.ispisi()}")
166. End Sub
```

Извршавање програма приказано је на слици 4.



Слика 4. Пример извршавања програма



Задатак: Модификовати претходни програм тако да се коришћењем методе *New* приликом креирања објекта иницијализују вредности за име и презиме.

New је метода која се покреће приликом креирања објекта. Може да прихвати произвољан број аргумената, или може се креирати без прослеђивања аргумената. Једном написана *New* метода од стране програмера пребрисаће методу која подразумевано постоји када се напише класа. Та метода је подразумевана и мора да постоји тако да иако се не напише приликом писања класе, *Visual Basic* ће је креирати. Метода нема повратни тип вредности и креира се коришћењем заглавља методе **Public Sub New**. За дати задатак метода је дата следећим кодом.

```
167. Public Sub New(ByVal ime As String, ByVal prezime As String)
168.     Me.ime = ime
169.     Me.prezime = prezime
170. End Sub
```

Главни програм је потребно изменити тако да се приликом иницијализације проследе вредности које ће иницијализовати објекат.

```
171. Sub Main()
```



```
172. Dim petar As Osoba
173. petar = New Osoba("Petar", "Petrović")
174. Console.WriteLine($"Ispis pozivom metode: {petar.ispisi()}")
175. End Sub
```

Сваки нови објекат заузима неки део оперативне меморије рачунара. Пошто рачунарска меморија није бесконачна, раније или касније, испуниће се подацима, а да добар део њих није више потребан. Објекти које се више неће користити требало би да буду избачени из оперативне меморије или чак уништени, односно врши се ослобађање меморије. То се може постићи на два начина:

1. помоћу експлицитне операције за брисање у *Visual Basic*-у се користи следећа наредба: **име променљиве = Nothing**,
2. помоћу имплицитног уклањања објекта кад он више није потребан ни једној променљивој или надређеном објекту (парент) у програмском окружењу.

Када се објекат класе изједначи са подразумеваном речју **Nothing**, аутоматски се покреће процедура **Finalize** унутар класе која се уништава, непосредно пре него што се она избрише из оперативне меморије. Ова процедура обично служи да експлицитно заустави или избрише покренуте процесе унутар класе.



Задатак: Модификовати претходни програм тако да се коришћењем методе *Finalize* приликом уништења објекта појави порука да је објекат уништен са временом када је уништен коришћењем *System.DateTime.Now* својства система.

Као и за методу *New* потребно је да се метода *Finalize* прегази. За разлику од *New* метода *Finalize* има другачији потпис па је приликом имплементације испоштовати њен потпис. Такође након извршавања кода који је додат, потребно је на крају додати линију *MyBase.Finalize()* која ће позвати оригиналну методу да објекат буде уништен.

```
176. Protected Overrides Sub Finalize()
177. Console.WriteLine($"Objekat uništen u vreme:
    {System.DateTime.Now}")
178. MyBase.Finalize()
179. End Sub
```

Са извршавањем програма појавиће се нова линија кода са исписом када је објекат уништен (слика 5).

```
C:\WINDOWS\system32\cmd.exe
Ispis pozivom metode: Petar Petrovic
Objekat unisten u vreme: 25/03/2018 21:41:23
Press any key to continue . . .
```



Слика 5. Пример извршавања програма

Наслеђивање класа

Наслеђивање (*inheritance*) класа смањује време потребно за развој програма, јер програмер стално и намерно позајмљује методе и особине објеката који већ постоје у систему.

Класа може истовремено наследити својства више других класа (вишеструко наслеђивање), а такође, може наследити више пута једну те исту наткласу (поновљено наслеђивање). Другим речима, структура програма има облик стабла са прекривањем грана и листова. Једнострукто наслеђивање води у хијерархије класа које имају строгу структуру стабла. Вишеструко наслеђивање води у мреже класа, а претраживање мрежа је обично вишезначно.



Задатак: Проширити претходни програм. Креирати нову класу *Student* која наслеђује класу *Osoba* и има атрибуте за број индекса и просек. Креирати објекат класе и исписати податке о атрибутима.

Резервисана реч за наслеђивање класе је ***Inherits***. Приликом имплементације резервисана реч мора бити испод имена надкласе а не у истом реду као што је код неких других програмских језика. Класа за овај пример дата је у наставку:

```
180. Public Class Student
181.     Inherits Osoba
182.     Public indeks As String
183.     Public prosek As Double
184.
185.     Public Sub New(ByVal ime As String, ByVal prezime As String,
        ByVal indeks As String, ByVal prosek As Double)
186.         MyBase.New(ime, prezime)
187.         Me.indeks = indeks
188.         Me.prosek = prosek
189.     End Sub
190.
191. End Class
```

На линији два декларише се наслеђивање, односно да је класа *Osoba* база класе *Student*. На линији 7 позива се метода *New* базне класе. Разлог за позив ове методе је што класа *Osoba* има своју методу *New*. Код објектно-оријентисаних језика правило је да уколико постоји метода за креирање објеката у основној класи, она се мора позвати у методи за креирање надкласе. Код Visual Basic-а ово се реализује позивом ***MyBase.New*** а затим се прослеђују аргументи базне класе.

Код главног програма потребно је изменити следећим кодом:



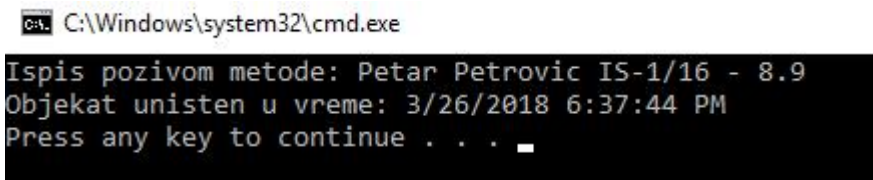
```
192. Sub Main()  
193.     Dim petar As Student  
194.     petar = New Student("Petar", "Petrović", "IS-1/16", 8.9)  
195.     Console.WriteLine($"Ispis pozivom metode: {petar.ispisi()}")  
196. End Sub
```

Покретањем програма добија се исти излаз као на слици 5. Потребно је креирати нову методу *podaci* која исписује податке о студенту. У класи *Student* потребно је додати следећи код:

```
197. Public Function podaci() As String  
198.     Return MyBase.ispisi() & $" {Me.indeks} - {Me.prosek}"  
199. End Function
```

У овој функцији коришћена је функција *ispisi* из базне класе а на њен резултат (који је текст) додат је текст са новим атрибутима.

У главном програму потребно је уместо методе *ispisi* позвати методу *podaci*. Извршавање примера дато је на слици 6.



```
C:\Windows\system32\cmd.exe  
Ispis pozivom metode: Petar Petrovic IS-1/16 - 8.9  
Objekat unisten u vreme: 3/26/2018 6:37:44 PM  
Press any key to continue . . .
```

Слика 6. Пример извршавања програма



БЕЛЕШКЕ



Динамички графички елементи

Динамичко креирање елемената

Сви елементи графичког корисничког интерфејса су у претходним примерима приручника постављани на форму апликације превлачењем у *designe modu* из палете са алаткама у време креирања програма. Међутим, може се јавити потреба да се елементи додају током извршавања програма, било као захтев корисника или захтев апликације, тада се елементи динамички убацују. Креирање графичког елемента се дефинише као:

Dim *ime grafičkog objekta* **As** *Tip Objekta* = **new** *Tip Objekta*()

Тип објекта је име графичког објекта које се налази у палети са алатима (Text Box, Button, Label...). Треба истаћи да декларацијом није креиран објекат класе, већ је само утврђено ког ће типа бити елемент, а све док се не креира он се не може користити. У наставку је дато неколико примера креирања објеката различитих графичких елемената.

```
200. Dim labela As Label = new Label()  
201. Dim dugme As Button = new Button()  
202. Dim polje As TextBox = new TextBox()
```

Декларацијом објекта планирају се ресурси за објекте, али садржај, величина објекта, као и родитеља (форма/панел) елемента нису дефинисани, треба водити рачуна да је неопходно поставити почетне вредности одређеним својствима објекта. Сва својства која нису иницијализована имају додељене подразумеване вредности које се налазе у класи објекта. Својства која би требало дефинисати су позиција елемента (подразумевано 0,0) димензије елемента (подразумевана вредност зависи од типа елемента) и ако елемент као компонента подржава текстуални приказ, одређени текст.

Следи пример постављања вредности неких атрибута дугмета:

```
203. Dim dugme As Button = New Button()  
204. dugme.Height = 40  
205. dugme.Width = 100  
206. dugme.Top = 10  
207. dugme.Left = 10  
208. dugme.Text = "Klikni me"
```

Претходно наведеним кодом су подешена нека основна својства дугмета. Али ако се покрене апликација, дугме неће бити видљиво. То се догађа зато што дугме није везано за ниједан контејнер елемент. Контејнер елемент може бити форма или елемент неки од елемената *GroupBox* или *Panel* који подржавају груписање елемената.



Да би елемент био додат у контејнер треба навести *imekontejner.Controls.Add* методу у чијем аргументу се наводи графичка контрола коју треба додати. За креирано дугме додавање у форму се реализује на следећи начин:

```
Me.Controls.Add(dugme)
```

Ако је потребно да се елемент који се додаје у контејнер, на пример у форму, појави при покретању апликације, онда се његово додавање пише се у **Form_Load** догађају.

Везивање догађаја за елементе

Претходном одељку објашњене су декларација и креирање графичких елемената, као и постављање атрибута тј. својстава графичких елемената у време извршавања програма. Сада ће бити речи о писању догађаја динамички креираних елемената. За дугме креирано у претходном примеру направити догађај такав да се појављује искачућа порука „Kliknuli ste na dugme“ када корисник кликне на дугме.

У претходним задацима и примерима креирани су постојећи догађаји, без да смо куцали код за њих. За почетак погледајмо како изгледа структура **Form_Load** догађаја.

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
```

О потпису рутине је у претходним поглављима било речи. *Private Sub Form1_Load* је рутина. Догађаји **јесу увек** рутине, али рутине **нису увек** догађаји. Аргументи ове рутине и оно што стоји после аргумената јесте оно што чини разлику између обичне рутине и догађаја. Први аргумент ове рутине **sender As Object** је објекат који је позвао овај догађај. Може бити нејасно зашто је *sender* декларисан као **Object** а не **Form**. Ово је типичан пример добрих страна објектног програмирања. Ако кажемо да је *sender* форма, онда овај догађај не може да се користи над дугметом, над лабелом или било којим другим графичким елементом.

Међутим, треба истаћи да не постоји никаква препрека да се догађај напише и на следећи начин:

```
Private Sub Form1_Load(sender As Form, e As EventArgs) Handles MyBase.Load
```

Програм би требало да ради апсолутно исто као што би радио и са *Object* елементом. Касније ће бити наведен пример везивања два различита елемента за један догађај и тада ће бити приказана предност коришћења *Object* елемента као првог аргумента догађаја.

Други аргумент **e As EventArgs** садржи информације о самом догађају (*EventArgs* је скраћено за *event arguments*). Какве могу бити информације о догађају? На пример ако је потребно добити координате где је корисник кликнуо



на елементу, оне се налазе у овој променљивој, информације о дугмету које је притиснуто на тастатури и друге информације, а може их бити доста у зависности од тога који је догађај позван (учитавање форме, клик миша итд.).

Остао је још део **Handles MyBase.Load** који неће бити детаљније објашњен, јер превазилази оквире овог курса.

Опционо:

Сваки елемент који се превлачи има своје готове догађаје, као и онај који се динамички креира. Ово који су превучени, односно користили смо GUI да их креирамо, у позадини заправо формирају како ће изгледати догађаји. Handles је веза између тих догађаја и онога што програмер додаје. На слици 1 може се видети да већ постоји догађај Load а да програмер у њега додаје код који ће се извршити.

```
657  ... Public Event Deactivate As EventHandler
662  ... Public Event FormClosing As FormClosingEventHandler
667  ... Public Event FormClosed As FormClosedEventHandler
672  ... Public Event Load As EventHandler
677  ... Public Event MdiChildActivate As EventHandler
683  ... Public Event MenuComplete As EventHandler
```

Слика 1. Предефинисани догађаји за Form1 објекат који се креира када креирамо Windows Forms Application

Креирање произвољних догађаја

Аргументи и Handles клауза **су опциони**. За задати пример који довољно је креирати тело рутине унутар форме.

```
209. Public Sub kliknutiNaDugme()
```

```
210.
```

```
211. End Sub
```

И направљен је догађај. Изгледа исто као и рутина. Свака рутина може постати догађај. Да би се то догодило потребно је повезати рутину са догађајем и елементом. Дефинисање повезивања дефинише се као:

AddHandler елемент.име догађаја, AddressOf име рутине која се извршава

Конкретно у задатом примеру везивање за елемент dugme и рутине kliknutiNaDugme дато је следећим кодом који треба написати у оквиру Form_Load догађаја:

AddHandler dugme.Click, AddressOf kliknutiNaDugme

Сада је пример решен, а у наставку следи комплетан програм решења.

```
212. Public Class Form1
```

```
213.     Private Sub Form1_Load(sender As Form, e As EventArgs)
        Handles MyBase.Load
```



```
214.         Dim dugme As Button = New Button()
215.         dugme.Height = 40
216.         dugme.Width = 100
217.         dugme.Top = 10
218.         dugme.Left = 10
219.         dugme.Text = "Klikni me"
220.         Me.Controls.Add(dugme)
221.         AddHandler dugme.Click, AddressOf kliknutiNaDugme
222.     End Sub
223.     Public Sub kliknutiNaDugme()
224.         MessageBox.Show("Kliknuli ste na dugme")
225.     End Sub
226. End Class
```

Следећи пример укључује рад са два елемента – дугметом и пољем за унос текста. Оба елемента биће креирана динамички. Текст из поља за унос текста треба приказати у искачућој поруци, и када корисник кликне на дугме, текст на дугмету треба да се промени у „Kliknuto“ и да се онемогући корисник да поново кликне на дугме.

У претходном примеру елемент је декларисан унутар Form_Load догађаја. Проблем са овим приступом је да се не може приступити елементу изван тог догађаја, а у овом примеру је потребно да је омогућен приступ елементима у оквиру догађаја који ће бити креирати. Тако да елементи бити декларисани у опсегу класе.

```
227. Public Class Form1
228.     Dim dugme As Button
229.     Dim txtPolje As TextBox
230. End Class
```

Иницијализација елеменат биће реализована у Form_Load догађају. У наставку дат је код са произвољним својствима елемената.

```
231. Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
232.     'Inicijalizacija tekstualnog polja
233.     txtPolje = New TextBox()
234.     txtPolje.Top = 10
235.     txtPolje.Left = 10
236.     txtPolje.Width = 150
237.     Me.Controls.Add(txtPolje)
238.
239.     'Inicijalizacija dugmeta
240.     dugme = New Button()
```



```
241.     dugme.Top = 40
242.     dugme.Left = 10
243.     dugme.Width = 150
244.     dugme.Height = 30
245.     dugme.Text = "Klikni me"
246.     Me.Controls.Add(dugme)
247. End Sub
```

Постоје два начина креирања догађаја који одговара критеријумима задатка. У наставку биће објашњена оба.

Први начин: Директан приступ елементима

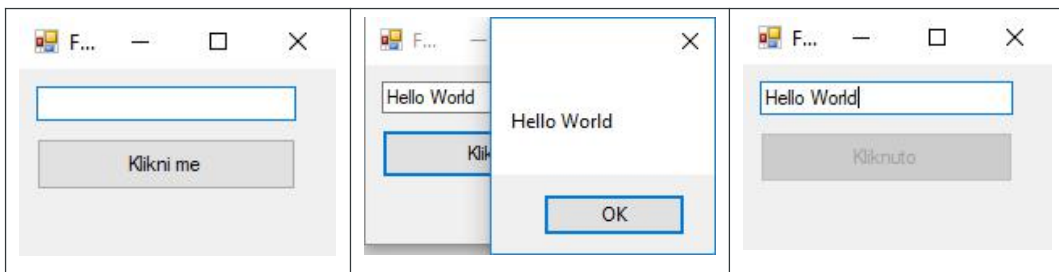
Први начин је најједноставнији и идеја је да се директно у догађају приступи елементима. Како су елементи глобални, можемо им приступити. Изглед догађаја дат је у наставку.

```
248. Public Sub klikPrviNacin()
249.     MessageBox.Show(txtPolje.Text)
250.     dugme.Text = "Kliknuto"
251.     dugme.Enabled = False
252. End Sub
```

Сада само треба у Form_Load догађају повезати рутину са догађајем дугмета.

AddHandler dugme.Click, AddressOf klikPrviNacin

Визуелни приказ извршавања кода дат је на слици 2.



Слика 2. Извршавање примера

Други начин: Приступ дугмету као sender објекту

Други начин обично се користити ако се приступа само датом елементу.

Заглавље рутине за аргументе имаће објекте sender и е без Handles дела кода. Заглавље рутине дато је следећим кодом:

```
Public Sub klikDrugiNacin(sender As Button, e As EventArgs)
```

Уместо Object у овом догађају за sender је постављено дугме. Ово је урађено зато што је из захтева задатка познато да ће само дугме користити овај догађај, тако да није потребно да буде дефинисан као Object аргумент. Тело рутине дато је следећим кодом:

```
253. Public Sub klikDrugiNacin(sender As Button, e As EventArgs)
```



```
254.     MessageBox.Show(txtPolje.Text)
255.     sender.Text = "Kliknuto"
256.     sender.Enabled = False
257. End Sub
```

Везивање рутине са догађајем исто је као и за први случај.



Везивање више елемената за један догађај

У претходном примеру објашњено је како се користи *sender* објекат за приступ објекту над којим се извршава догађај. Дилеме није било, само један елемент је могао да изазове догађај тако да могли смо да му приступимо директно. У реалним апликацијама, може се јавити потреба да више елемената позива исти догађај.

На пример игрица Minesweeper (слика 3).



Слика 3. Пример игрице Minesweeper

Ова игрица је представљена као матрица димензија 10x10 где корисник може да кликне на свако дугме. Кад кликне на дугме отвориће се поље, бомба или ако је поље празно, сва адјуктивно везана поља у околини.

Када корисник кликне на неко поље (дугме) рецимо 6,3, било би редундантно да програм испитује све позиције од 1,1 до 6,3 и да испитује да ли је то дугме кликнуто на основу на пример позиције миша. Нема потребе за тим када постоји бољи начин, а то је да се од догађаја добије информација који га је елемент позвао. Том елементу се може приступити кроз *sender* објекат односно све информације о томе елементу као што су позиција, боја позадине, текст итд. су доступне кроз њега (*sender* објекат).



Задатак: Написати програм који креира матрицу дугмади димензија 3x3 која заузима целу форму апликације. Када корисник кликне на неко дугме боја дугмета треба да постане наранџаста.

Све делове програма осим догађаја за клик дугмета треба писати у оквиру *Form_Load* догађаја. На почетку треба декларисати матрицу, декларисати и променљиве за ширину и висину дугмета који се могу одмах иницијализовати:

```
258. Dim dugmad(2, 2) As Button
```

```
259. Dim sirinaDugmeta As Integer = Me.ClientSize.Width / 3
```



```
260. Dim visinaDugmeta As Integer = Me.ClientSize.Height / 3
```

Димензије сваког дугмета израчунавају се на основу димензија контејнера. Треба водити рачуна да уколико би у линији 3 било наведено *Me.Height* у димензије би биле урачунате и навигација и мени трака форме. *ClientSize* је својство које даје димензије контејнера форме, без околних ивица и трака.

Пре инстанцирања дугмади треба написати догађај који ће се реализовати када се кликне на неко дугме.

```
261. Public Sub klikNaDugme(sender As Button, e As EventArgs)
```

```
262.     sender.BackColor = Color.Orange
```

```
263. End Sub
```

Сада када је дефинисана матрица дугмади потребно је свако дугме инстанцирати и поставити на форму. Без обзира што су елементи матрице типа *Button*, а не неког простог типа (*integer*, *double*,...) поступак приступа сваком елементу матрице је исти. За сваки елемент матрице врши се креирање објекта, постављање димензија дугмета, одређивање позиције дугмета на форми и повезује се претходно написан догађај.

```
264. For i As Integer = 0 To 2
```

```
265.     For j As Integer = 0 To 2
```

```
266.         dugmad(i, j) = New Button()
```

```
267.         dugmad(i, j).Width = sirinaDugmeta
```

```
268.         dugmad(i, j).Height = visinaDugmeta
```

```
269.         dugmad(i, j).Top = j * visinaDugmeta
```

```
270.         dugmad(i, j).Left = i * sirinaDugmeta
```

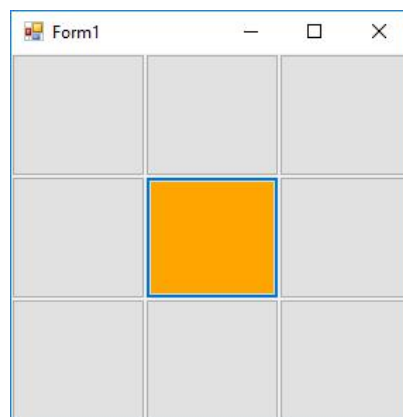
```
271.         AddHandler dugmad(i, j).Click, AddressOf klikNaDugme
```

```
272.         Me.Controls.Add(dugmad(i, j))
```

```
273.     Next
```

```
274. Next
```

Када се покрене програм требало би да се добије форма слична форми на слици 4.





Слика 4. Пример форме са динамички креираним дугмадима

Задатак за вежбу: Модификовати претходни пример тако да тако да матрица буде реда 5x5 и се да уместо унапред дефинисане боје, када се кликне на дугме оно обоји у случајно изабрану боју.



БЕЛЕШКЕ



Рад са ресурсима и датотекама

Све апликације на неком нивоу чувају подешавања или податке корисника. Ти подаци се могу чувати у меморији програма, текстуалној датотеци, word документу, бази података или у неком другом формату. Најосновнији начин чувања података је смештање у неку променљиву. Овај начин складиштења података је примењиван у претходним вежбама, али његова примена није адекватна уколико податке треба чувати и након завршетка рада програма.

Датотека је скуп података који се меморише на рачунару у његовој сталној меморији (хард диск) или на неком од спољних медија двд, цд... Програми су такође датотеке које складиште код који је преведен како би могао да се извршава са рачунара. Екстензије датотека дају јаснију одредницу о намени датотека (ехе је датотека са извршним кодом, txt је текстуална датотека итд.).

Осим екстензија датотеке имају и своје путање. То је алијас за адресу у меморији рачунара где се налази запис датотеке. Корисник то види као текстуалну путању. Коришћењем путање корисник може да креира датотеку, да је измени, прочита њен садржај или да је обрише. Када се ради са датотекама које заузимају већи меморијски простор (нпр. видео снимци високе резолуције), корисник мора да буде упознат са ресурсима свог рачунара.

Приступ ресурсима рачунара

Почев од стандарда 2010 језика Visual Basic уведена је резервисана реч **My** која дозвољава брз приступ најчешће коришћеним референцама, методама и функцијама апликације. Коришћењем **My** резервисане речи може се приступити неком од објеката као **My.Application** који дозвољава манипулацију директно свим члановима креираног пројекта, **My.Computer** која приказује податке везане за рачунар и **My.User** која дозвољава приступ привилегијама корисника који је улоган на рачунар.

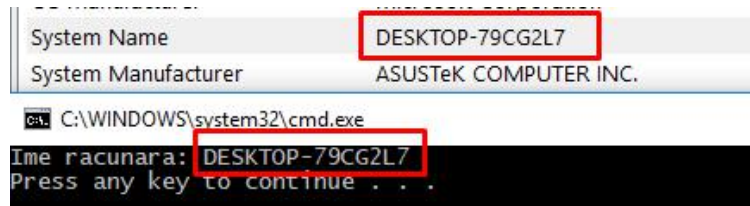
My.Computer

My.Computer има неколико објеката који пружају информације о корисниковом рачунару и манипулацију над неким његовим ресурсима. Неки од објеката су Audio, Clipboard, Network, Info i FileSystem. У наставку биће демонстрирана поједина својства:

Прво својство је **Name** које враћа име корисниковог рачунара. За коришћење овог својства у коду је потребно написати наредбу **My.Computer.Name**. Позивом следеће линије кода

```
Console.WriteLine("Име рачунара: " & My.Computer.Name)
```

при покретању апликације у конзоли се приказује испис као на слици 1.



Слика 1. Приказ имена рачунара у конзоли и у *System information* палети

Info објекат враћа информације о оперативном систему (назив и верзија), доступном физичком и виртуелном простору. Следећи код исписаће основне информације о рачунару и његовом оперативном систему:

```
275. Console.WriteLine("Ime računara: " & My.Computer.Name)
276. Console.WriteLine("Operativni sistem: " &
    My.Computer.Info.OSFullName)
277. Console.WriteLine("Verzija: " & My.Computer.Info.OSVersion)
278. Console.WriteLine("Slobodna fizička memorija: " &
    My.Computer.Info.TotalPhysicalMemory)
279. Console.WriteLine("Slobodna virtuelna memorija: " &
    My.Computer.Info.TotalVirtualMemory)
```

My.Computer.FileSystem објекат

FileSystem објекат пружа информације о ресурсима рачунара који се односе на његове партиције, директоријуме и различите типове датотека. Његово својство **CurrentDirectory** приказује путању директоријума из којег је покренута апликација. Имплементацијом следећег кода

```
Console.WriteLine("Trenutni direktorijum: " &
    My.Computer.FileSystem.CurrentDirectory)
```

добија се испис путање из које је покренута апликација:

```
Trenutni direktorijum: M:\Programming\Visual Basic\Datoteke i
izuzeci\Resursi\bin\Debug
```

Путања примера разликоваће се у зависности од тога са ког је рачунара покренута. Уколико је програм покренут у *Debug* режиму путања до апликације завршиће се са *bin\Debug*. То је тренутна локација где је *Visual Studio* генерисао ехе датотеку и пропратне датотеке потребне да би програм радио.

Debug директоријум не треба користити за крајњу апликацију. Када је апликација завршена и спремна за дистрибуцију, користи се *Build* опција и *Release* ражим за креирање апликације. Генерисани програм се може преузети из директоријума *bin\Release* за дистрибуцију корисницима.

Постоје програми који кориснику нуде опцију да изабере хард диск (нпр. приликом инсталације оперативног система). Својство **Drives** садржи листу објеката **DriveInfo**. *DriveInfo* објекат садржи информације о партицији као што су:

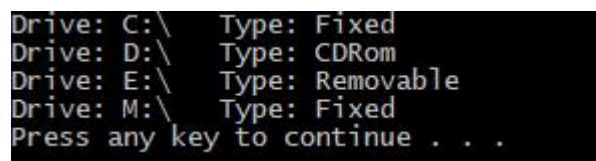


- 🐾 *AvailableFreeSpace* – простор који је доступан на датој партицији изражен у byte-овима.
- 🐾 *DriveType* – да ли је партиција хард диск, CD ром, мрежни диск или неки други тип.
- 🐾 *Name* – враћа име партиције
- 🐾 *TotalSize* – укупан простор на диску (заузет и слободан) изражен у byte-овима.

Следећи код коришћењем *For Each* петље исписује партиције рачунара:

```
280. Dim FS = My.Computer.FileSystem
281. For Each particija In FS.Drives
282.     Console.WriteLine($"Drive: {particija.Name}    Type:
        {particija.DriveType}")
283. Next
```

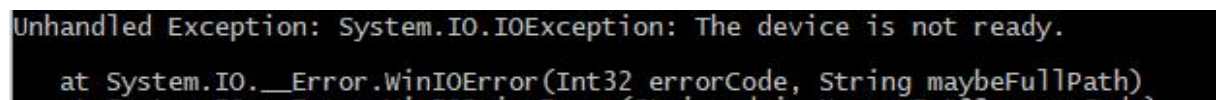
У првој линији дефинисана је променљива *FS* која нема тип податка. Употреба променљиве на овај начин зове се алијас и имплицитно јој је додељен тип вредности *My.Computer.FileSystem* како би се користио краћи назив. У зависности од рачунара добиће се излаз у конзоли као на слици 2.



```
Drive: C:\    Type: Fixed
Drive: D:\    Type: CDRom
Drive: E:\    Type: Removable
Drive: M:\    Type: Fixed
Press any key to continue . . .
```

Слика 2. Испис листе доступних партиција

Уколико се дода и *TotalSize* као параметар за испис а постоји *CD/DVD* медијум без убаченог диск појавиће се грешка као на слици 3.



```
Unhandled Exception: System.IO.IOException: The device is not ready.
at System.IO.__Error.WinIOError(Int32 errorCode, String maybeFullPath)
at System.IO.DriveInfo.GetDriveType(String driveName, Int32 errorCode)
```

Слика 3. Пријава грешке уколико није убачен диск у *CD/DVD* читач

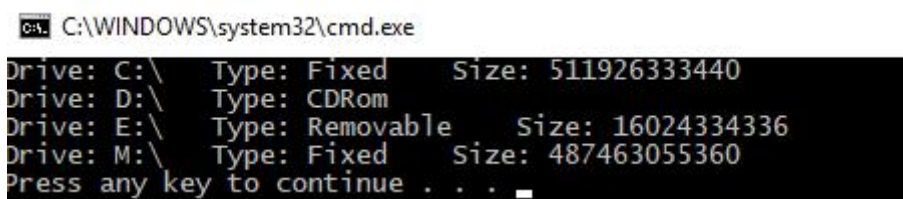
Овај проблем се решава провером да ли је партиција *CD/DVD* и да ли је у њу убачен диск. Следећи код сматра се напреднијим градивом и препоручује се студентима које детаљније занима рад са партицијама.

```
284. Dim FS = My.Computer.FileSystem
285. For Each particija In FS.Drives
286.     Dim showSizeInfo As Boolean = True
287.     If particija.DriveType = IO.DriveType.CDRom Then
288.         Dim cdrom As New IO.DriveInfo(particija.Name)
289.         If Not cdrom.IsReady Then
290.             showSizeInfo = False
291.         End If
```



```
292.     End If
293.
294.     Console.WriteLine($"Drive: {particija.Name}    Type:
        {particija.DriveType}")
295.     If showSizeInfo = True Then
296.         Console.WriteLine($"    Size: {particija.TotalSize}")
297.     End If
298.     Console.WriteLine()
299. Next
```

У 4 линији пореди се вредност поља *DriveType* са елементима листе *IO.DriveType* (*Fixed*, *Removable*, *CDRom*). Уколико је вредност *CDRom* креира се нови објекат класе *DriveInfo* која садржи својство *IsReady* које враћа информацију да ли је *CD* убачен у медијум. Провером вредности својства које враћа *True* или *False* у конзоли ће се написати излаз који је сличан излазу на слици 4.



Слика 4. Приказ доступних партиција, њихових типова и доступних величина

FileSystem класа има велики број метода које се могу користити у апликацији. У табели 1 дате су методе које се учестало користе у апликацијама које раде са датотекама.

DirectoryExists (<i>putanja</i>)	Провера да ли постоји директоријум за наведену путању.
CreateDirectory (<i>putanja</i>)	Креира нови директоријум за наведену путању.
CopyDirectory (<i>stara putanja, nova putanja, pregazi</i>)	Копира стари директоријум на нову локацију. Уколико је вредност <i>pregazi</i> постављена на <i>true</i> , ако већ постоји директоријум са истим именом и датотекама које постоје на старој путањи, те датотеке биће замењене датотекама из старе путање.
DeleteDirectory (<i>putanja, režim</i>)	Брише директоријум за наведену путању. Режим за брисање може бити



<i>brisanja</i>)	DeleteDirectoryOption.DeleteAllContents која брише сав садржај у директоријуму и DeleteDirectoryOption.ThrowIfDirectoryNonEmpty која јавља грешку уколико директоријум није празан.
MoveDirectory (<i>stara putanja, nova putanja, pregazi</i>)	Премешта стари директоријум на нову локацију. Уколико је вредност <i>pregazi</i> постављена на true, ако већ постоји директоријум са истим именом и датотекама које постоје на старој путањи, те датотеке биће замењене датотекама из старе путање.
RenameDirectory (<i>putanja, novi naziv</i>)	Мења назив директоријума
GetDirectories (<i>putanja</i>)	Враћа листу свих поддиректоријума на наведеној путањи
FileExists (<i>putanja</i>)	Провера да ли постоји датотека за наведену путању.
CopyFile (<i>stara putanja, nova putanja, pregazi</i>)	Копира датотеку са старе на нову локацију. Уколико је вредност <i>pregazi</i> постављена на true, ако већ постоји датотека са истим именом, биће замењена новом датотеком.
DeleteFile (<i>putanja</i>)	Брише датотеку на наведеној путањи
MoveFile (<i>stara putanja, nova putanja, pregazi</i>)	Премешта стару датотеку на нову локацију. Уколико је вредност <i>pregazi</i> постављена на true, ако већ постоји датотека са истим именом, биће замењена новом датотеком.
RenameFile (<i>putanja, novi naziv</i>)	Мења назив датотеке.
GetFiles (<i>putanja</i>)	Враћа листу свих датотека које се налазе на наведеној путањи
ReadAllText (<i>putanja</i>)	Чита сав текстуални садржај из датотеке
WriteAllText (<i>putanja, tekst, append</i>)	Уписује текстуални садржај у датотеку. Уколико је вредност <i>append</i> опције постављена на true, садржај се додаје на крај датотеке, у супротном стари садржај се мења новим.

Табела 1. Учестало коришћење методе класе *FileSystem*

**Задатак:** Креирати форму као на слици.

The screenshot shows a standard Windows application window titled 'Form1'. Inside the window, there is a label 'Putanja' above a single-line text input field. Below this is a label 'Sadržaj' above a multi-line text area. At the bottom of the form, there are two buttons: 'Snimi sadržaj' on the left and 'Pročitaj sadržaj' on the right.

Апликација може да прочита и да сними садржај у зависности од путање коју је унео корисник. Пре извршавања било које од функционалности, проверава се да ли датотека постоји и уколико не постоји, кориснику се приказује одговарајућа искачућа порука. Сваки садржај који се снима, брише претходни садржај датотеке.

Пре имплементације догађаја за снимање и учитавање садржаја, у структури форме биће имплементирана метода *postojiDatoteka* која проверава да ли постоји датотека и враћа вредност *True* уколико постоји, а уколико не постоји јавља одговарајућу поруку кориснику и враћа вредност *False*. Кôд за методу је:

```
300.     Private Function postojiDatoteka(ByVal putanja As String)
        As Boolean
301.         If My.Computer.FileSystem.FileExists(putanja) Then
302.             Return True
303.         Else
304.             MessageBox.Show("Datoteka za zadatu putanju ne
                postoji!")
305.             Return False
306.         End If
307.     End Function
```

Метода ће бити позивана у оквиру догађаја за снимање и читање садржаја. Коришћењем метода *ReadAllText* и *WriteAllText* може се прочитати и снимити садржај датотеке. Кôд за догађаје дат је у наставку.

Кôд за догађај снимања садржаја:



```
308.     Private Sub btnSnimi_Click(sender As Object, e As
        EventArgs) Handles btnSnimi.Click
309.         If postojiDatoteka(txtPutanja.Text) Then
310.
            My.Computer.FileSystem.WriteAllText(txtPutanja.Text,
            txtPrikaz.Text, False)
311.         End If
312.     End Sub
```

Кôд за догађај читања садржаја:

```
313.     Private Sub btnProcitaj_Click(sender As Object, e As
        EventArgs) Handles btnProcitaj.Click
314.         If postojiDatoteka(txtPutanja.Text) Then
315.             txtPrikaz.Text =
                My.Computer.FileSystem.ReadAllText(txtPutanja.Text)
316.         End If
317.     End Sub
```

Читање и упис садржаја у датотеке секвенцијално

Метода *ReadAllText* прочитаће сав текстуални садржај. Постоје случајеви када корисник не жели да види сав садржај датотеке, можда апликација треба да врати одређени део датотеке за неку претрагу. Датотеке се могу читати секвенцијално, и постоје различите класе и методе које ће у зависности од захтева корисника враћати садржај датотеке. У оквиру овог поглавља класа која ће бити коришћена је ***StreamReader*** која дозвољава секвенцијално читање садржаја линију по линију или карактер по карактер. Демонстрације и описи односиће се на читање садржаја ред по ред.

StreamReader класа има методу ***ReadLine*** која чита карактере све док не дође до знака *Enter* или краја датотеке, и прочитане карактере враћа као *String*. Проблем са овом методом је да се чита само један ред датотеке. Уколико датотека има више редова, само први биће прочитан док остали редови биће занемарени. Коришћењем петље *While* и методе *ReadLine* могуће прочитати све редове. Када дође до краја датотеке *ReadLine* вратиће вредност *Nothing*.



Задатак: Креирати форму као на слици.



Апликација може да прочита садржај у зависности од путање коју је унео корисник. Садржај се чита ред по ред и уколико је ред празан, испис тог реда се прескаче.

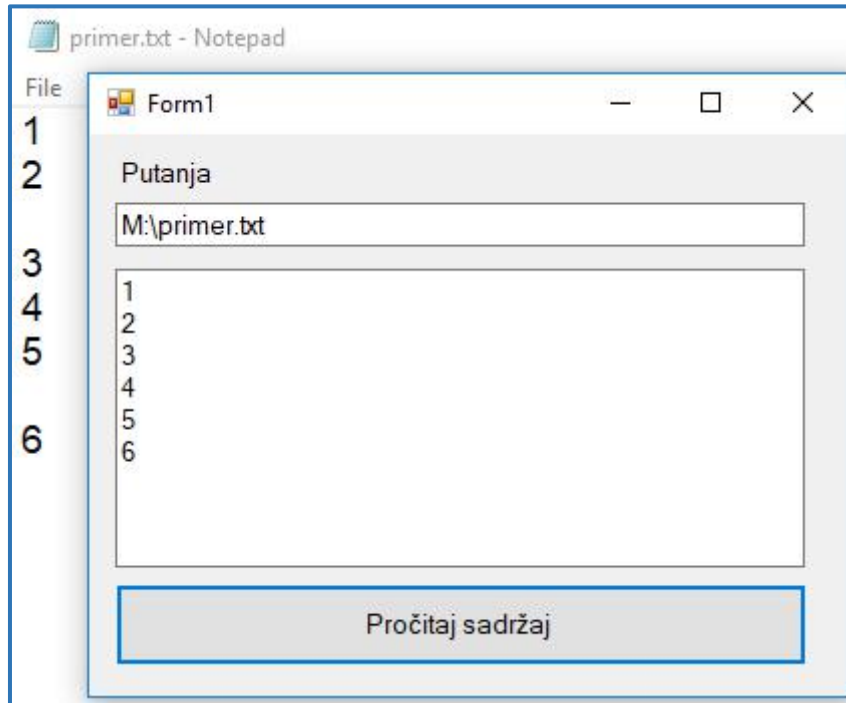
```
318. Private Sub btnProcitaj_Click(sender As Object, e As EventArgs)
      Handles btnProcitaj.Click
319.     Dim citac As System.IO.StreamReader = New
        System.IO.StreamReader(txtPutanja.Text)
320.     Dim linija As String
321.     While True
322.         linija = citac.ReadLine()
323.         If IsNothing(linija) Then
324.             Exit While
325.         End If
326.         If linija.Length = 0 Then
327.             Continue While
328.         End If
329.
330.         txtPrikaz.Text &= linija & vbCrLf
331.     End While
332.     citac.Close()
333. End Sub
```

Класа *StreamReader* припада библиотеци *System.IO* због чега је потребно да се при коришћењу користи пун назив класе (библиотека и име класе) или да се увезе библиотека. Када је остварена комуникација са датотеком отворен коришћењем бесконачне *while* петље садржај се чита све док линија не врати вредност *Nothing*. Са функцијом ***IsNothing*** испитује се да ли променљива има



вредност *Nothing*. На крају да би се датотека могла користити потребно је ослободити датотеку затворити. Објекат *citac* када заврши са читањем треба позвати методу **Close** и затворити датотеку.

Извршавање примера дато је на слици 5.



Слика 6. Пример извршавања програма

Секвенцијални упис садржаја у датотеку реализује се коришћењем класе **StreamWriter** из библиотеке *System.IO*. Упис садржаја се може имплементирати коришћењем метода **Write** и **WriteLine**. Као и код *StreamReader*-а након уписа садржаја потребно је затворити датотеку. Уколико се садржај уписује у датотеку која не постоји, она ће бити креирана.



Задатак: Написати апликацију која произвољан низ имена уписује у текстуалну датотеку *imena.txt*. Свако име налази се у једном реду.

```
334. Sub Main()  
335.     Dim imena() As String = {"Ana", "Jelena", "Svetlana",  
    "Dusan", "Andreja"}  
336.     Dim pisac As System.IO.StreamWriter = New  
    System.IO.StreamWriter("M:\imena.txt")  
337.     For Each ime As String In imena  
338.         pisac.WriteLine(ime)  
339.     Next  
340.     pisac.Close()
```



341. `End Sub`

У овом примеру коришћена је *For Each* петља за пролаз кроз низ имена, након чега се свако име уписује у датотеку. Метода *WriteLine* додаје карактер за *Enter* након садржаја тако да је свако име у новом реду.



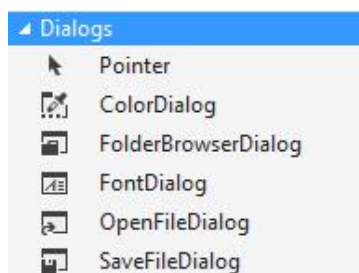
БЕЛЕШКЕ



Дијалог прозори, Менији и MDI апликације

Дијалог прозори

У претходним поглављима приказан је рад са једном врстом искачућих прозора, а то су *Message* прозори који поред могућности да се испишу одређене поруке, омогућавају одређену врсту интеракције, тј. одабир неке опције (Yes, No, Cancel). Постоји још пет дијалога који програмеру нуде једноставну реализацију комуникације са корисником коме се на једноставан начин нуде различита подешавања: боје, фонта, меморисања датотека, отварања датотека итд. Ови дијалози се могу изабрати из Toolbar палете у секцији „Dialogs“ (слика 1).



Слика 1. „Dialogs“ секција

ColorDialog

ColorDialog прозор омогућује кориснику из палете боја (слика 2) изабере одређену боју и да је сними као вредност.



Слика 2. „ColorDialog“ прозор

За отварање дијалога користи се наредба „ShowDialog“ која отвара прозор дијалога. Избором боје у својство „Color“ дијалога смешта се изабрана боја. Уколико корисник притисне дугме „Cancel“ у својству се налази подразумевана црна боја. Вредности својства су објекти типа „Color“.



Задатак 1: Креирати форму са дугметом које отвара ColorDialog и у зависности од изабране боје мења позадину форме.



Из секције „Dialogs“ палете алата треба превући елемент „ColorDialog“ и дати му име „dialogColor“, а дугмету дати назив „btnBirajBoju“.

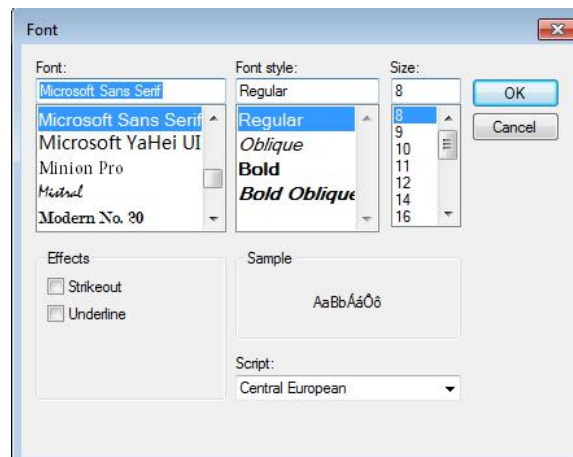
Кôд за решење задатка је дат у наставку:

```
342. PrivateSub btnBirajBoju_Click(sender AsObject, e AsEventArgs)  
    Handles btnBirajBoju.Click  
343.     dialogColor.ShowDialog()  
344. Me.BackColor = dialogColor.Color  
345. EndSub
```

У линији 2 користи се метода „ShowDialog“. Ова метода отвара искачући дијалог, након чега се од корисника чека акција. Боја коју корисник изабере додељује се својству „BackColor“ које ће као резултат променити боју позадине.

FontDialog

Дијалог за промену фонта омогућава промену типа, величине и стила фонта. Као и за „ColorDialog“ методом „ShowDialog“ приказује се дијалог на слици 3.

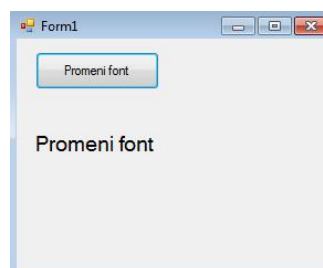


Слика 3. Избор својстава фонта коришћењем FontDialog

Коришћењем својства „Font“ могу се добити подаци из дијалога које је корисник изабрао.



Задатак 2: Креирати форму дату на:



Кликом на дугме „Promeni font“ потребно је променити својства лабеле са текстом „Promeni font“ коришћењем дијалога за промену фонта.

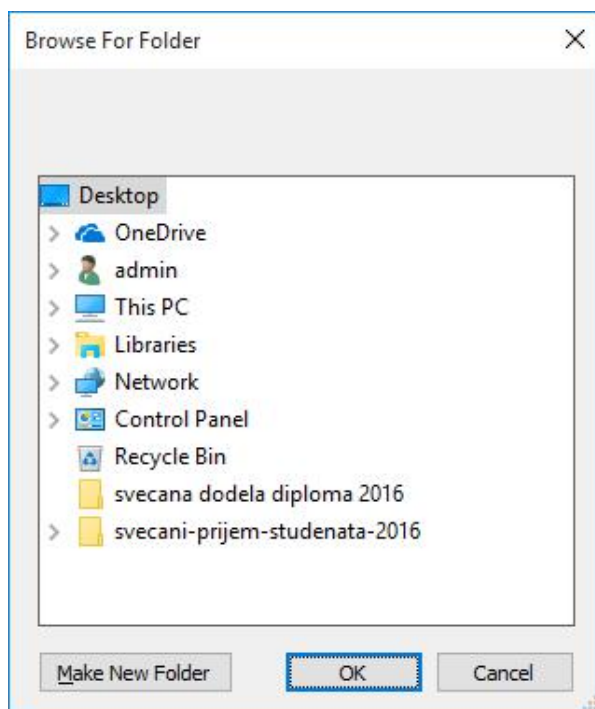


Из секције „Dialogs“ палете алата треба превући елемент „FontDialog“ и дати му име „dialogFont“, дугмету дати назив „btnPromeniFont“, а лабели назив „lblFontPromena“. Кôд за решење задатка је дат у наставку:

```
346. PrivateSub btnPromeniFont_Click(sender AsObject, e AsEventArgs)  
    Handles btnPromeniFont.Click  
347.     dialogFont.ShowDialog()  
348.     lblFontPromena.Font = dialogFont.Font  
349. EndSub
```

FolderBrowserDialog

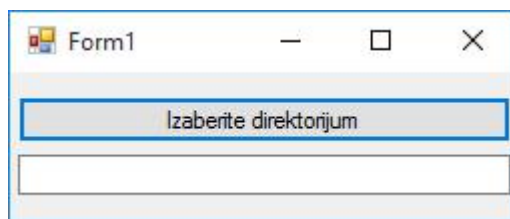
FolderBrowserDialog је дијалог прозор за избор директоријума (слика 5). Уколико корисник не изабере одређени директоријум путања која се враћа кориснику дијалога је празан стринг. Са атрибутом „SelectedPath“ добија се путања директоријума који је корисник изабрао.



Слика 4. Претрага рачунара коришћењем FolderBrowserDialog-а



Задатак 3: Креирати форму дату на слици:



Кликом на дугме „Izaberite direktorijum“ потребно је приказати путању



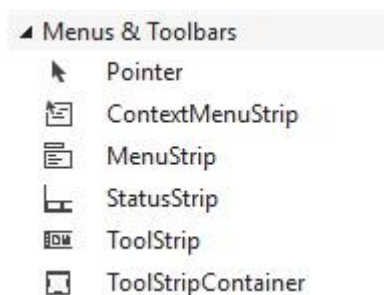
изабраног директоријума у текстуалном пољу испод дугмета.

Из секције „Dialogs“ палете алата на форму треба превући елемент „FolderBrowserDialog“ и промении му својство Name на „dialogFolderBrowser“, дугмету дати назив „btnBrowseFolder“, а текстуалном пољу назив „txtFolderPath“. Код за решење задатка је дат у наставку:

```
350. PrivateSub btnBrowseFolder_Click(sender AsObject, e AsEventArgs)  
    Handles btnBrowseFolder.Click  
351.        dialogFolderBrowser.ShowDialog()  
352.        txtFolderPath.Text = dialogFolderBrowser.SelectedPath  
353. EndSub
```

Менији

Свака Windows апликација има неки вид навигације којом корисник има интеракцију са апликацијом. У Visual Basic-у уз помоћ развојног окружења могуће је на једноставан начин креирати меније апликације. Сви типови менија доступни су у Toolbar палети у секцији „Menus & Toolbars“ (слика 1).



Слика 1. „Menus & Toolbars“ секција

MenuStrip мени

Први мени који ће бити презентован је „MenuStrip“. „MenuStrip“ је мени за главну навигацију апликације. Прво треба убацити мени на форму. Кад је то урађено, појавиће се поље за унос елемената у мени (слика 2). Приликом уношења елемента, појавиће се могућност да унесете следећи елемент са десне стране или нови елемент који је подмени ставке која се уноси са менија.



Слика 2. Унос елемената у мени навигацију

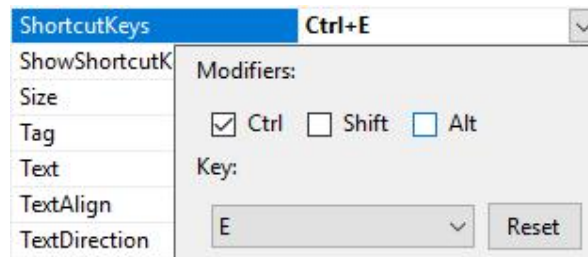
Пре додавања елемената у мени, потребно му је дати одговарајуће име. У пољу својства (Name) унети вредност mnuGlavni. Конвенција у именовању објеката је да мени елементи имају префикс mni испред имена менија.

У прво поље треба унети текст „Апликација“. Кликом на креирани елемент у пољу својства (Name) унети вредност „miАпликација“. Овај елемент менија имаће



функционалност родитељског елемента који ће садржати друге елементе менија.

У поље испод менија „Апликација“ унети нови елемент „Затвори“, а у пољу својства (Name) вредност „miZatvori“. Пре имплементације функционалности елемента, треба додати могућност да се коришћењем пречице активира његова функционалност. У листи својстава треба пронаћи својство „ShortcutKeys“ и ставити као пречицу команду „Ctrl+E“, што се остварује уношењем слова Е и потврђивањем поља за потврду код Ctrl (слика 3).



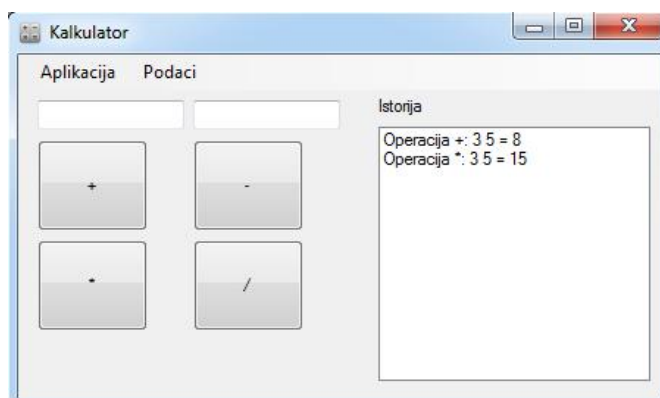
Слика 3. Имплементација пречице за елемент менија

Следећи корак је додавање функционалности елементу. У „Events“ секцији „Properties“ панела пронаћи „Click“ догађај и назвати га „zatvaranjeAplikacije“. У догађај унети следећи код:

```
354. PrivateSub zatvaranjeAplikacije(sender AsObject, e AsEventArgs)  
    Handles miZatvori.Click  
355. Me.Close()  
356. EndSub
```



Задатак: Креирати дизајн као на слици.



У главном менију постоје два подменија: „Апликација“ „Подаци“. Подмени „Апликација“ садржи опцију „Изађи“ (Ctrl + X). Подмени „Подаци“ садржи опције „Сачувај“ (Ctrl + S) и „Обриши“ (Ctrl + D).

Опција „Изађи“ затвара апликацију. Опција „Сачувај“ омогућава кориснику да садржај листе дате испод лабеле „Историја“ сачува на жељеној локацији као текстуалну датотеку коришћењем дијалог прозора за чување датотека. Опција „Обриши“ омогућава кориснику да обрише садржај листе.



Реализовати клик догађаје дугмади „+“, „-“, „*“ и „/“ тако да се у зависности од кликнутог дугмета изврши одговарајућа математичка операција за бројеве које корисник унесе у текстуална поља.

Апликација треба да има одговарајућу иконицу као и наслов „Kalkulator“.

Пре навођења решења, потребно је усаглашавање са конвенцијама именовања. Главни мени назвати „mnuGlavni“, његове подменије „miAplikacija“ и „miPodaci“, а њихове ставке „milzadji“, „miSacuvaj“ и „miObrisi“. За друге визуелне елементе конвенције су следеће: текстуална поља – „txtOp1“ и „txtOp2“; дугмад – „btnPlus“, „btnMlnus“, „btnPut“ и „btnDeli“; лабела – „lblIstorija“ и листа „lstIstorija“.

Први елементи за који треба креирати догађај је „milzadji“. Догађај ће бити затварање апликације и биће назван „zatvoriAplikaciju“. Кôд за реализацију овог догађаја дат је у наставку:

```
357. PrivateSub zatvoriAplikaciju(sender AsObject, e AsEventArgs)  
    Handles miIzadji.Click  
358. Me.Close()  
359. EndSub
```

Следећи елемент за који треба креирати догађај је „miSacuvaj“. Потребно је креирати догађај који снима датотеку и назвати га „sacuvajDatoteku“. Кôд за програмирање овог догађаја следи :

```
360. PrivateSub sacuvajDatoteku(sender AsObject, e AsEventArgs)  
    Handles miSacuvaj.Click  
361. Dim saveDatotekaDialog AsNewSaveFileDialog()  
362.     saveDatotekaDialog.Filter = "Tekstualna datoteka|*.txt"  
363.     saveDatotekaDialog.Title = "Snimi tekstualnu datoteku"  
364.     saveDatotekaDialog.ShowDialog()  
365. If saveDatotekaDialog.FileName <>""Then  
366. Dim file As System.IO.StreamWriter = New  
    IO.StreamWriter(saveDatotekaDialog.OpenFile())  
367. Dim tekstZaUpis AsString = ""  
368. ForEach item In lstIstorija.Items  
369.     tekstZaUpis += tekstZaUpis + item.ToString &  
    vbCrLf  
370. Next  
371.     file.WriteLine(tekstZaUpis)  
372.     file.Close()  
373. EndIf  
374. EndSub
```

Следећи елемент за који треба креирати догађај је „miObrisi“. Креирати догађај „obrisiPodatke“. Кôд за догађај дат је у наставку:



```
375. PrivateSub obrisiPodatke(sender AsObject, e AsEventArgs) Handles  
    miObrisi.Click  
376.    lstIstorija.Items.Clear()  
377. EndSub
```

Пре догађаја за операције потребно је декларисати променљиве „op1“, „op2“ и „rezultat“ као глобалне променљиве типа Double.

```
378. Public ClassForm1  
379. Dim op1, op2, rezultatAs Double
```

Реализација догађаја „saber“ дата је у наставку. За преостале догађаје сами имплементирајте код.

```
380. PrivateSub saberi(sender AsObject, e AsEventArgs) Handles  
    btnPlus.Click  
381.    op1 = Val(txtOp1.Text)  
382.    op2 = Val(txtOp2.Text)  
383.    rezultat = op1 + op2  
384.    lstIstorija.Items.Add($"Operacija +: {op1}{op2} =  
        {rezultat}")  
385. EndSub
```

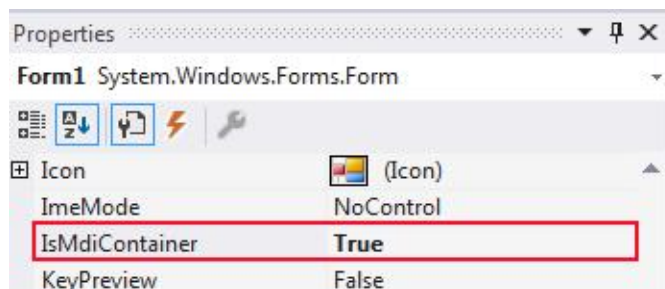
MDI апликације

The multiple-document interface (MDI) дозвољава програмерима да креирају апликације које садрже више форми. Апликације као што Microsoft Excel, Microsoft Word, и сам Visual Studio су пример MDI апликација.

MDI апликација дозвољава кориснику да прикаже више прозора истовремено, од којих је сваки прозор независан од другог. Документи или деца прозори се налазе унутар родитељског прозора. Ово је суштинска разлика између отварања новог прозора позивом клика на дугме и прозора који ће бити унутар документа.

Креирање MDI форме

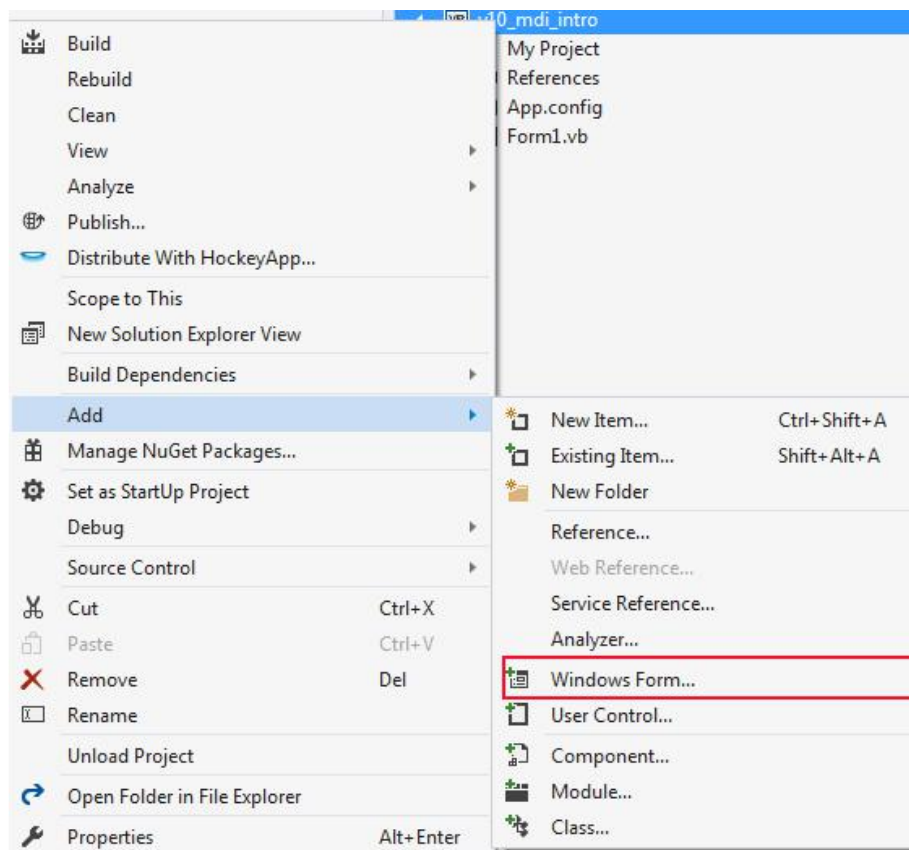
Да бисте креирали MDI апликацију потребно је изабрати форму која ће бити родитељ / контејнер елементима. Затим у палети за својства треба пронаћи опцију „IsMdiContainer“ и поставите њену на „true“ (слика x). Тело форме промениће боју, односно појавиће се сиви контејнер.





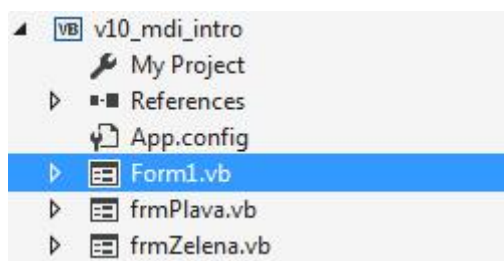
Слика х. Опција „IsMdiContainer“

Када је креиран родитељ, потребно је креирати и децу форме што се остварује десним кликом миша на име пројекта. У менију пронаћи опцију „Add“, а затим у понуђеној листи опција изабрати „Windows Form...“ (слика х).



Слика х. Избор елемента форме

Напрвити две форме „frmPlava“ и „frmZelena“. Прва форма имаће плаву боју позадине, друга ће имати зелену боју позадине. Новокреиране форме појавиће се у пројекту (слика х).



Слика х. Нове форме у пројекту

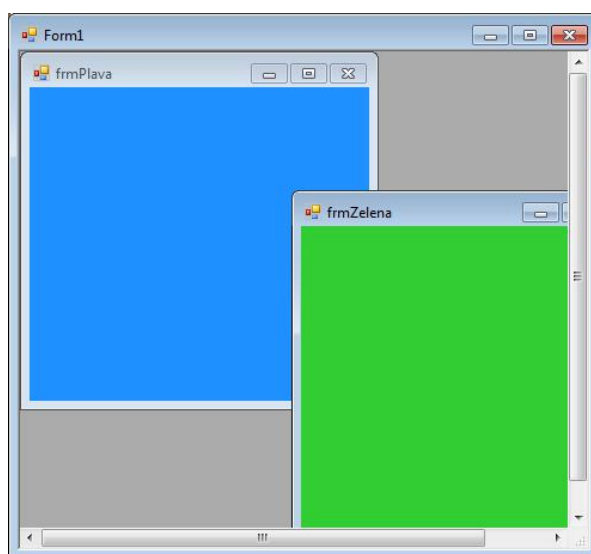
Када су форме креиране оне могу бити позване у родитељском елементу. Потребно је креирати објекте за дате форме и доделити им родитеља. У „Load“ догађају унети следећи код:

```
386. PrivateSub Form1_Load(sender AsObject, e AsEventArgs)  
    Handles MyBase.Load
```



```
387. Dim frm1 AsNewfrmPlava
388. Dim frm2 AsNewfrmZelena
389.     frm1.MdiParent = Me
390.     frm2.MdiParent = Me
391.     frm1.Show()
392.     frm2.Show()
393. EndSub
```

На линијама 4 и 5 користи се својство „**MdiParent**“ повезује форме frm1 и frm2 са формом Form1 тако што Form1 постаје њихов родитељ. Да би форме биле видљиве потребно их је приказати, тј. искористи методу „**Show**“ (линије 6 и 7). На крају се добија изглед сличан слици x.



Слика x. Пример MDI форме

Креираном апликацијом је практично приказано како изгледа MDI апликација. У наставку следи једна практична примена MID-и апликације.

Креирање MDI апликације



Задатак: Креирати апликацију за упис заинтересованих кандидата у Високу школе електротехнике и рачунарства.

Инструкције за решење задатка:

Када се апликација покрене кориснику апликације се приказује образац за унос кандидата. Поља која треба да корисник попуни су:

- ❖ ЈМБГ кандидата
- ❖ Име и презиме кандидата
- ❖ Датум рођења
- ❖ Место пребивалишта
- ❖ Жељени смер кандидата (смерови су понуђени у падајућој



листи)

❖ Контакт телефон кандидата

Кликом на дугме „Сачувај“ креира се нова датотека која се зове „jmbg.kd“ где је ЈМБГ име датотеке, а „kd“ екстензија датотеке. Сви записи се чувају у директоријуму „kandidati“. Путања је подразумевано „C:\kandidati“.

Поред прозора за унос постоји и прозор за приказ кандидата где је могуће видети листу свих унетих кандидата.

Решење задатка:

Потребно је креирати апликацију која има две форме „frmUnosKandidata“ и „frmPrikazKandidata“ и главну форму за апликацију.

Родитељској форми треба дати фиксне димензије. У палети са својствима потребно је изменити следећа својства:

- ❖ Size – 800, 600
- ❖ MaximizeBox – False
- ❖ MinimizeBox – False
- ❖ MaximumSize – 800,600
- ❖ MinimumSize – 800,600

Овим је обезбеђено да креирана апликација има фиксну величину 800x600 пиксела. Следећи елемент је форма „frmUnosKandidata“ над којим ће бити извршене измене. Исте измене је потребно извршити и над формом „frmPrikazKandidata“:

Својство „FormBorderStyle“ потребно је поставити на вредност „None“.

- ❖ Својство „Size“ треба променити у „800, 600“,
- ❖ Својство „ControlBox“ на вредност „False“
- ❖ Својство „MaximizeBox“ на вредност „False“
- ❖ Својство „MinimizeBox“ на вредност „False“
- ❖ Својство „WindowState“ треба променити у „Maximized“.

Пре везивања форме „frmUnosKandidata“ са главном формом, треба декларисати и иницијализовати обе форме. Код је дат у наставку:

```
394. Public Class Form1
395. Dim frmUnos As New frmUnosKandidata
396. Dim frmPrikaz As New frmPrikazKandidata
397. End Class
```

Овај приступ решењу је примењен јер ће се ове форме смењивати када корисник у менију изабере који ће прозор желети да прикаже. Да би се подразумевано приказала форма „frmUnosKandidata“ у „Load“ догађају треба позвати методу „Show“. Такође, у истом догађају за обе форме треба својству



„MdiParent“ доделити вредност „Me“ како би форме везали са главном формом. Кôд за догађај дат је у наставку:

```
398. PrivateSub Form1_Load(sender AsObject, e AsEventArgs)  
    Handles MyBase.Load  
399.     frmUnos.MdiParent = Me  
400.     frmPrikaz.MdiParent = Me  
401.     frmUnos.Show()  
402. EndSub
```

Када се покрене апликација приказаће се форма у којој ће се отворити форма за унос података ученика. Како на форми нема никаквих елемената потребно је направити форму за унос. У том циљу треба креирати форму са дизајном приказаном на слици x.

Слика x. Дизајн форме за унос кандидата

У клик догађају дугмета снимити треба унети следећи кôд којим се реализује унос кандидата:

```
403. PrivateSub btnSnimiKandidata_Click(sender AsObject, e  
    AsEventArgs) Handles btnSnimiKandidata.Click  
404. My.Computer.FileSystem.WriteAllText($"C:\kandidati\{txtJmbg.Text}  
    .kd", $"  
405. JMBG: {txtJmbg.Text}  
406. Ime i prezime: {txtImePrezime.Text}  
407. Datum rođenja: {dpDatumRodjenja.Value.ToString()}  
408. Mesto prebivališta: {txtMestoPrebivalista.Text}  
409. Željeni smer: {cbZeljeniSmer.SelectedItem.ToString()})
```



```
410. ".Trim(), False)
411. MessageBox.Show($"Kandidat {txtJmbg.Text} је успешно snimljen",
    "Успешно snimljen kandidat")
412. EndSub
```

Ово решење подразумева да постоји већ директоријум под називом „kandidati“. За самостални рад се препоручује модификација наведеног решења тако да се пре уписа података провери да ли постоји директоријум на путањи „C:\kandidati“ и уколико не постоји потребно је креирати га.



Креирати датотеке са најмање пет кандидата у датом директоријуму.

Следећи корак је креирање форме за приказ кандидата. Пре креирања дизајна форме, потребно је прво испланирати на који начин ће подаци о кандидатима бити приказани тј. како обрадити податке за приказ и како најефективније написати код.

Сви подаци о кандидатима доступни су у текстуалним датотекама са екстензијом „kd“. Да би добили податке о кандидатима потребно је:

- ❖ Прочитати имена свих доступних датотека из директоријума и сачувати их у низу
- ❖ Проћи кроз низ имена датотека и на основу имена и путање отворити сваку датотеку
- ❖ Из отворене датотеке прочитати садржај и обрадити податке
- ❖ Из обрађених података узети потребне податке и приказати их у листи.

Сада када је дефинисано како треба приказати кандидате, може се приступити решењу. Креирати форма као на слици х.

Слика х. Дизајн за листу за приказ кандидата.



Најбољи начин да се реализује скуп корака који су наведени је да се за сваку ставку направи посебна метода. Прва методе+а: „Прочитати имена свих доступних датотека из директоријума и сачувати их у низу“. Постоје две операције „Прочитати имена датотека“ и „сачувати их у низу“. У претходним поглављима објашњен је рад са датотекама и показано је како се чита садржај директоријума. Питање које остаје односи се на низ. Ако би низ декларисали као локалну променљиву, он не би био употребљив касније. Тако да у оквиру форме за приказ кандидата треба декларисати глобалну променљиву за низ имена датотека. Кôд је дат у наставку:

```
413. PublicClass frmPrikazKandidata
414. Dim nizImenaDatoteka() AsString
415. EndClass
```

Уместо писања методе која би прочитала низ имена датотека користиће се већ постојећа библиотека „System.IO“ у циљу читања низа имена датотека. Из ове библиотеке користи се наредба „Directory.GetFiles“ која ће из дате путање прочитати имена свих датотека и вратити њихова имена као низ.

Где написати овај кôд? Ако погледамо функционалност, листа кандидата треба да буде учитана када се кориснику прикаже форма, дакле у догађају „Load“ биће имплементиран следећи кôд:

```
416. Imports System.IO
417. PublicClass frmPrikazKandidata
418. Dim nizImenaDatoteka() AsString
419.
420. PrivateSub frmPrikazKandidata_Load(sender AsObject, e
    AsEventArgs) Handles MyBase.Load
421.     nizImenaDatoteka = Directory.GetFiles("C:\kandidati")
422. EndSub
423. EndClass
```

У овој фази писања програма није креирана могућност у апликацији да се тестира написани кôд. Лоша је пракса да се без провере настави даље са развојем апликације. Тестирање форме независно од извршавања MDI апликације биће објашњено у наредном тексту.

Први корак је коришћење „debugger“ алата у циљу праћења стања променљиве, односно провере да ли је позвана метода вратила очекиване резултате. На месту које одговара линији 7, на сивој линији са леве стране треба притиснути леви тастер миша. Појавиће се црвена тачка која означава тачку прекида извршавања програма (break point) као на слици x. Тачке прекида се постављају у циљу провере тачности одређених делова кода. На тај начин, пратећи део по део извршавања програма може се детектовати у ком делу програма се налази грешка, ако програм не ради оно што је програмер желео.



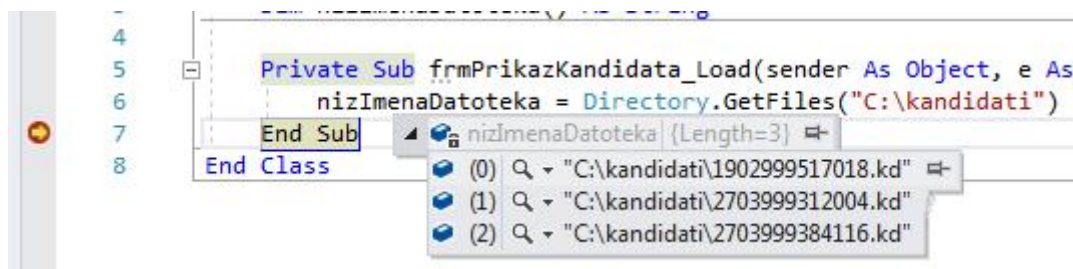
```
1 Imports System.IO
2 Public Class frmPrikazKandidata
3     Dim nizImenaDatoteka() As String
4
5     Private Sub frmPrikazKandidata_Load(sender As Object, e As EventArgs) Handles MyBase.Load
6         nizImenaDatoteka = Directory.GetFiles("C:\kandidati")
7     End Sub
8 End Class
```

Слика х. Коришћење „debugger“ алата

Следећи корак је измена у родитељској форми унети следећи код:

```
424. PrivateSub Form1_Load(sender AsObject, e AsEventArgs)
      HandlesMyBase.Load
425.     frmUnos.MdiParent = Me
426.     frmPrikaz.MdiParent = Me
427. frmPrikaz.Show()
428. EndSub
```

По покретању апликације, неће се отворити форма већ ће се програм зауставити са извршавањем на седмој линији кода а излаз који се добија дат је на слици х.



```
4 Private Sub frmPrikazKandidata_Load(sender As Object, e As EventArgs) Handles MyBase.Load
5     nizImenaDatoteka = Directory.GetFiles("C:\kandidati")
6 End Sub
7
8 End Class
```

nizImenaDatoteka {Length=3}

- (0) Q - "C:\kandidati\1902999517018.kd"
- (1) Q - "C:\kandidati\2703999312004.kd"
- (2) Q - "C:\kandidati\2703999384116.kd"

Слика х. Приказ заустављеног извршавања програма

Сада када је проверено да су успешно учитана имена датотека потребно је отворити сваку датотеку и прочитати њен садржај. Пре реализације читања текстуалне датотеке потребно је одлучити како ће садржај датотеке бити приказан. Најбољи начин би био да у листи имамо ред са подацима:

ЈМБГ – Име и презиме – Место – Жељени смер

Ово јесте компликованији приступ, лакше је једноставно прочитати цео текст, али је доста боље, јер дохватање ових података појединачно омогућава да се по потреби креирају и објекти класа (нпр. ажурирање корисника захтева рад са објектима). Због прегледности кода у класу ће бити додата нова методу која обрађује садржај текстуалне датотеке, тј. чита датотеку ред по ред и издваја потребне информације.

```
429. PrivateFunction procitajSadrzaj(putanjaDatoteke AsString)
      AsString
430.
```

431. **EndFunction**

Метода прихвата као аргумент путању до текстуалне датотеке, а као повратну вредност враћа форматиран текст за листу кандидата. У наставку је дат део методе који ће бити касније објашњен:

```
432. PrivateFunction procitajSadrzaj(putanjaDatoteke AsString)
    AsString
433. Dim jmbg, imeprezime, mesto, smer AsString
434. Dim sr AsStreamReader = NewStreamReader(putanjaDatoteke)
435. DoWhile sr.Peek() >= 0
436. Dim pocitanaLinija AsString = sr.ReadLine()
437. If pocitanaLinija.Contains("JMBG:") Then
438.     jmbg = pocitanaLinija.Split(": ")(1)
439. EndIf
440. Loop
441. sr.Close()
442. Return"{jmbg} - {imeprezime} - {mesto} - {smer}"
443. EndFunction
```

Класа StreamReader је представљена у претходним поглављима тако да је читање појединачних линија већ објашњено. На линијама 6,7 и 8 је If наредба која проверава да ли се у прочитаној линији налази део текста „JMBG:“. Уколико постоји тај део текста потребно је поделити учитану линију са сепаратором „: “. Могуће је у сепаратору не користити размак и позвати „trim“ методу када потребни податак буде извучен.

Структура датотеке је таква да је свака линија датотеке дата у формату „**Назив податка: вредност**“. Поделом датог стринга на индексу 0 налази се назив податка (JMBG, Ime i prezime, Datum рођења, Mesto prebivališta, Željeni smer), а на индексу 1, њихове вредности. Потребне су само вредности. Линије 6, 7 и 8 узимају JMBG кандидата. Решење издвајања имена и презимена, места и смера реализовати самостално.



Креирати преостале услове за податке – име и презиме, место и жељени смер.

Сада је потребно изменити методу „Load“ и додати петљу у којој ће бити прочитане информације о кандидату и тај садржај ће бити приказан у листи. Код је дат у наставку:

```
444. PrivateSub frmPrikazKandidata_Load(sender AsObject, e
    AsEventArgs) Handles MyBase.Load
445.     nizImenaDatoteka = Directory.GetFiles("C:\kandidati")
446. ForEach putanja AsStringIn nizImenaDatoteka
447. Dim infoKandidata AsString = procitajSadrzaj(putanja)
448.     lstKandidati.Items.Add(infoKandidata)
```



449. [Next](#)

450. [EndSub](#)



Задаци за самостални рад

Задатак 1. Креирати дизајн као на слици:

Апликација креира конфигурационе датотеке (.conf) у којима се налазе подешавања за боју позадине, апликације, боју текста, избора изгледа и величине текста. Корисник може да изабере где ће да сними датотеку и под којим називом. Све измене које корисник изабере могу се видети у панелу са стране (боја позадине панела, боја текста лабеле у панелу, тип текста лабеле у панелу). Дугмад која раде са дијалозима за боју и сама мењају боју позадине.

Задатак 2. Дobili сте задатак за креирање апликације која ради са формирањем рачуна за корисника за куповину. Корисник апликације када покрене апликацију добија екран са листом рачуна који су креирани. Уколико нема рачуна у директоријуму, у листи стоји порука да нема рачуна. Рачуни се налазе на предефинисаној путањи „C:\гасипи“а екстензија сваког рачуна је „.rc“.

Апликација нуди и опцију формирања новог рачуна. Постоји дете форма која у себи има елемент за избор датума, вишелинијско текстуално поље за унос артикала као и поље за унос цене. Сваки артикал се чува у формату dd-MM-uuuu-NN:mm.rc.

Апликација има мени који поред могућности да корисник отвори форме за унос и приказ има и опције да корисник затвори апликацију, и операцију да обрише све рачуне из датог директоријума.



БЕЛЕШКЕ



Мултимедијални елементи

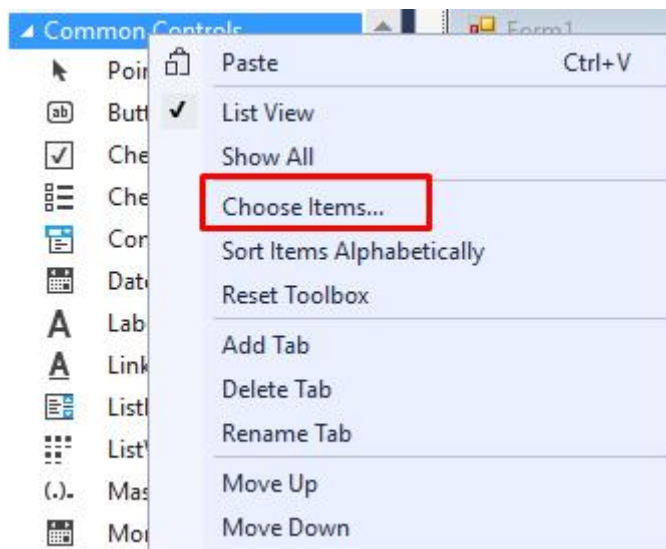
Данашње апликације често имају неке мултимедијалне садржаје. Без обзира да ли су у питању рекламе, видео материјали или позадинска музика, апликација поред функционалности треба кориснику да пружи и визуелно/звучно пропратно искуство. Писање сопствених компоненти које поседују овакве функционалности захтева добро познавање језика и његових библиотека и често захтева и писање сложених блокова програма да би се реализовале ове функционалности.

Уколико није потребна специјална функционалност која се односи на мултимедијалне садржаје, могу се користити већ постојеће компоненте које се могу додати у апликацију. Предност је уштеда у писању кода јер већина ових компоненти има једноставну синтаксу за реализацију потребних функционалности, али неке од њих имају неке недостатке. У наставку су представљене компоненте које се баве аудио/видео садржајем, приказом веб страница и приказом PDF докумената.

Windows Media Player

Један од начина да се убаце видео и аудио датотеке је коришћењем елемента који уграђује Windows Media Player као графички елемент у апликацију.

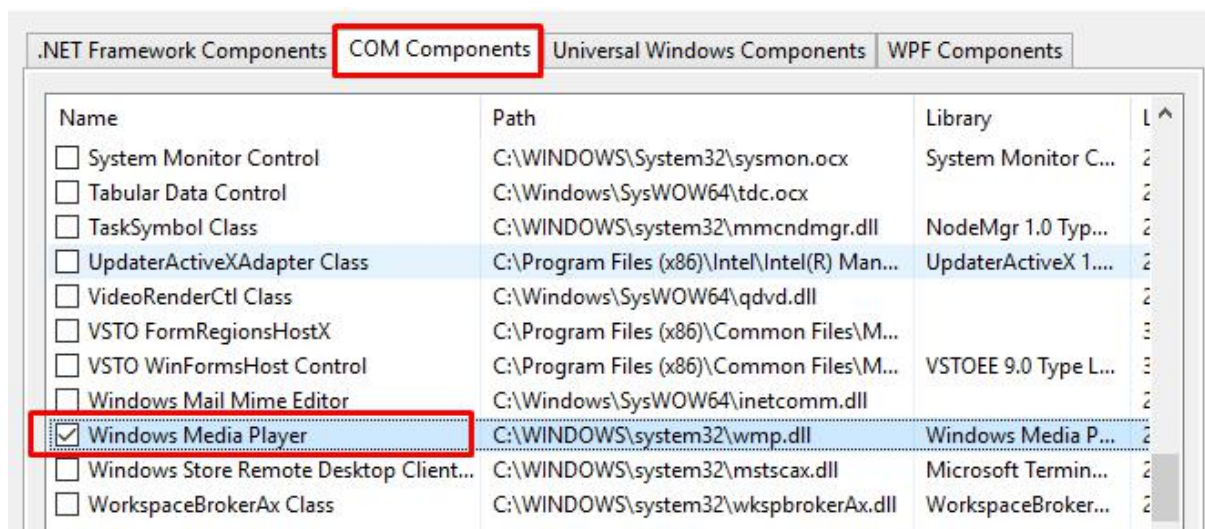
Веома често у Toolbox-у нема понуђеног елемента „Windows Media Player“ (на неким окружењима је могуће да елемент постоји), па га треба додати. Да би додали Windows Media Player у Toolbox, потребно је активирати десно дугме миша унутар палете на неки од заглавља (нпр. Common Controls), а затим изаберати опцију „Choose items...“ (слика 1).



Слика 1. Избор опције „Choose Items...“

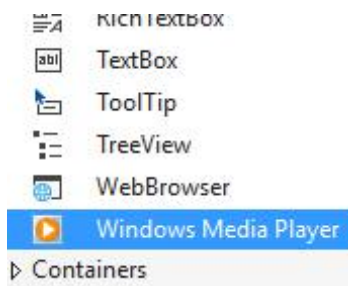


Отвориће се палета која нуди избор разних компоненти које се могу додати. Windows Media Player се налази у „COM Components“ табу (картици). На овој картици треба пронаћи одговарајућу компоненту у листи и потврдити поље поред имена (слика 2).



Слика 2. Додавање „Windows Media Player“ компоненте у Toolbox

У падајућој листи требало би да се појави Windows Media Player елемент (слика 3).



Слика 3. Windows Media Player компонента „Common Tools“ листи

По привлачењу компоненте на форму, треба је развући да попуни целу форму. Да би пустили аудио или видео снимак потребно је променити својство „URL“ и додати путању на рачунару где се налази датотека са мултимедијалним садржајем (нпр. C:\Users\Dusan\Videos\The Cinematic Orchestra - To Build A Home (Live at Sydney).mp4). Кад се покрене апликација, као и код нормалног Windows Media Player-а, очекује се да се активирање дугмета „Play“ и требало би после тога да видео снимак почне да се приказује.

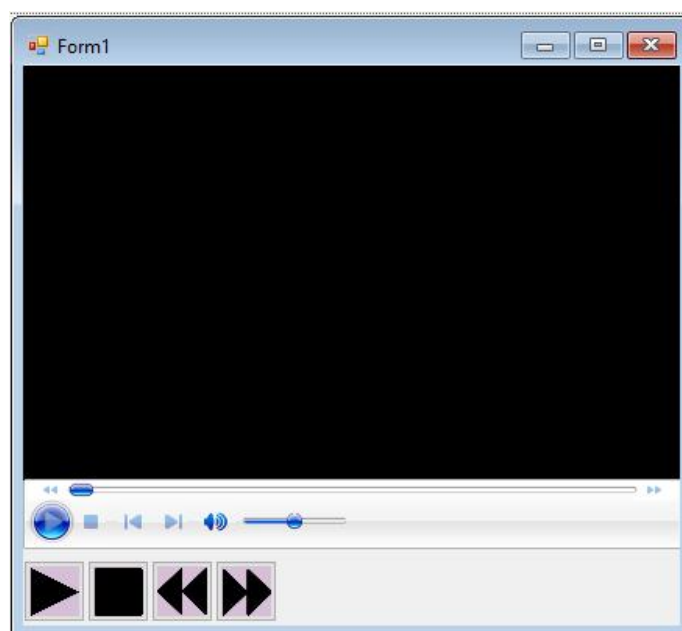


Слика 4. Демонстрација Windows Media Player на покренутој апликацији

На овај начин је реализовано елементарно убацивање мултимедијалних садржаја, али често је потребно да видео или аудио садржаји буду презентовани у неком дизајну који је у складу са стилем апликације (нпр. да буду дугмад црне боје итд.). Акције овог елемента се могу покретати коришћењем других графичких елемената.

Наредни задатак је креирање видео player-а.

За почетак треба изменити дизајн пројекта као што је приказано на слици 5.



Слика 5. Дизајн player-а



Пре него писања кода потребно је елементима дати следеће називе - Windows Media player биће axWMP, а називи дугмади: btnPlay, btnStop, btnRewind, btnForward респективно.

Затим треба изабрати player и подесити следећа својства:

- enableContextMenu - Својство које приказује мени на десни клик миша на видео - Ово својство је потребно поставити на вредност „false“.
- uiMode - дефинише графичке компоненте player-а (play button, stop итд.) - Овом својству је потребно навести вредност „none“.

У клик догађају дугмета „Play“ треба имплементирати приказ видео датотека. Име за догађај је „playVideo“. У догађају се наводи само једна линија кода:

```
451. Private Sub playVideo(sender As Object, e As EventArgs) Handles  
    btnPlay.Click  
452.     axWMP.Ctlcontrols.play()  
453. End Sub
```

Компонентама player –а може се приступити коришћењем Ctlcontrols, а са методом play се активирају функционалности play дугмета. Када се покрене апликација видео садржај се аутоматски приказује. Ово се дешава зато што се autostart својство активира ако постоји URL својство. То значи да се URL не може дефинисати директно кроз panel својстава. Зато треба обрисати вредност својства из палете својстава. URL својство би требало дефинисати када корисник кликне на дугме „Play“. Измени клик догађај дугмета додавањем линије кода:

```
454. Private Sub playVideo(sender As Object, e As EventArgs) Handles  
    btnPlay.Click  
455.     axWMP.URL = "C:\Users\Dusan\Downloads\The Cinematic  
    Orchestra - To Build A Home (Live at Syndey).mp4"  
456.     axWMP.Ctlcontrols.play()  
457. End Sub
```

Сада би требало да се садржај приказује тек када корисник активира дугме „play“. За вежбу самостално имплементирати код за заустављање приказивања видеа. Ову акцију треба реализовати као догађај клик дугмета „btnStop“.

Остаје да се дода функционалност дугмадима „Forward“ и „Rewind“. Њихова функционалност се имплементира када корисник има више видео датотека. За демонстрацију ових дугмади биће коришћени предефинисани низ видеа. Изменити код форме на следећи начин:

```
458. Public Class Form1  
459.     Dim nizVideoDatoteka() As String = {  
460.         "C:\Users\Dusan\Downloads\The Cinematic Orchestra - To  
        Build A Home (Live at Syndey).mp4",  
461.         "C:\Users\Dusan\Downloads\Thrice - Black Honey.mp4",  
462.         "C:\Users\Dusan\Downloads\Tool - Sober.mp4"
```



```
463.     }
464.     Dim currentIndex As Integer = 0
465.
466.     Private Sub playVideo(sender As Object, e As EventArgs)
         Handles btnPlay.Click
467.         axWMP.URL = nizVideoDatoteka(currentIndex)
468.         axWMP.Ctlcontrols.play()
469.     End Sub
470. End Class
```

Прво ће бити имплементиран код за дугме Forward. Променљива „currentIndex“ ће бити искоришћена за пуштање текућег видеа. На почетку је вредност ове променљиве 0 односно, активирањем дугмета „Play“ читаваће се први елемент низа. Дугме Forward има исту функционалност као и дугме play, с тим што пре пуштања видеа проверава да ли може да пусти следећи видео. Ако може, онда ће следећи видео бити пуштен. Креирати нови догађај за клик миша под називом „forwardToNextVideo“.

```
471. Private Sub forwardToNextVideo(sender As Object, e As EventArgs) Handles btnForward.Click
472.     If currentIndex < nizVideoDatoteka.Length - 1 Then
473.         currentIndex += 1
474.         axWMP.URL = nizVideoDatoteka(currentIndex)
475.         axWMP.Ctlcontrols.play()
476.     Else
477.         MessageBox.Show("There isn't next video available!")
478.     End If
479. End Sub
```

У линији 2 кода је наредба гранања која проверава да ли је тренутна позиција у оквиру низа. Уколико јесте, индекс низа се повећава за један и приказује се следећи видео. Ако не постоји следећи елемент низа (дошло се до краја низа), путем поруке која се приказује у искачућем прозору, обавештава се корисник да није могуће учитати следећи видео. Реализацију функционалност дугмета backward потребно је урадити за самосталну вежбу.

Picture box

Слике често представљају саставни део апликације. Графички елемент који се користи за приказ слика је „PictureBox“. Неке од функционалности овог елемента су помињане у другом поглављу, а сада ће бити приказана и нека напреднија својства.

До сада је коришћен Import локалних датотека, оодносно у приказаним примерима даване су апсолутне путање до датотека које су биле коришћене, као на пример „C:\Documents\slika.png“. Проблем се појављује ако се апликација пребаци на други рачунар, онда овако задата путања не мора бити тачна, па слика неће бити учитана јер датотека у којој је слика смештена које пронађена.



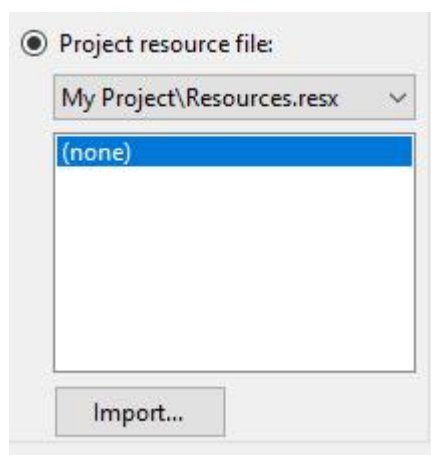
Овај проблем се може решити креирањем директоријума Resources у оквиру пројекта. У овај директоријум треба сместити све ресурсе пројекта; слике, текстуалне датотеке, аудио и видео садржаје, ако их има итд.

Кликом десног дугмета миша да пројекат, изабрати опцију „Add“, а затим опцију „New Folder“ и креирати директоријум „Resources“. Име директоријума мора бити „Resources“. Требало би да се у пројекту налази директоријум као што је приказано на слици 6.



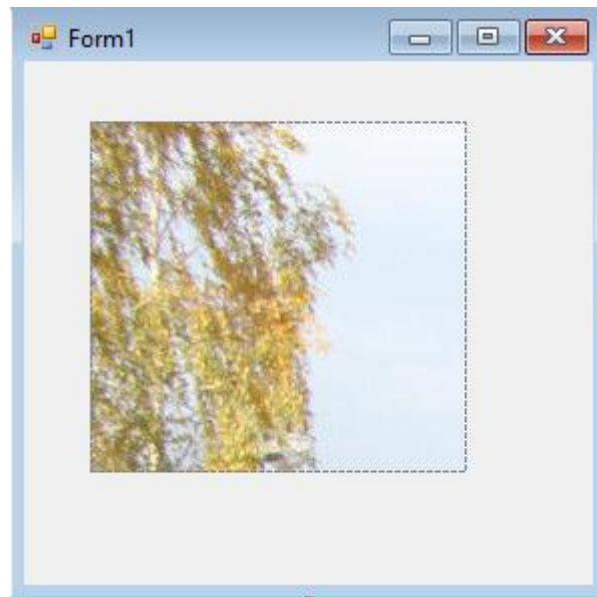
Слика 6. Resources директоријум у оквиру пројекта

Додати PictureBox елемент у пројекат. Изабрати својство „Image“. Требало би да осим својства „Local Resource“ буде понуђено и „Project resource file“ (слика 7). Активирањем дугмета „Import...“ слика се чува као ресурс.



Слика 7. Project resources опција увоза слике

Уколико је у фолдер Resources раније додата нека слика онда листа на слици 7 неће бити празна. Поступак додавања је исти као и за локални ресурс, разлика је у томе што ће Visual Studio направити копију ресурса и сместити га у креирани директоријум. Када је ресурс додат, он ће бити додат у листу и може бити изабран за приказивање слика 8.



Слика 8. Додата слика ресурса у Picture Box елементу

Један од проблема који може настати је неусклађеност величине слике и PictureBox елемента. На пример на слици 8 није приказан аутомобил који се налази на слици, јер је приказан само део слике. Да би цела слика била приказана, потребно је подесити својство `SizeMode` PictureBox елемента, које има подразумевану вредност „Normal“. Вредности „SizeMode“ својства могу бити:

- `Normal` - подразумевана вредност; слика је постављена у горњем левом углу и уколико је ширина или дужина већа од контејнера слика ће бити одсечена у димензијама контејнера.
- `StretchImage` - слика ће се проширити или скупити тако да стане у оквиру контејнера слике. Уколико пропорције контејнера и величине слике нису исте, слика ће бити изобличена (може изгледати сабијено).
- `Zoom` - слика ће се скупити тако да стане у оквиру контејнера слике. Коришћењем `zoom`-а за разлику од `StretchImage` слика ће попунити простор по висини или ширини PictureBox елемента али се неће развући по другој димензији, већ ће задржати своје пропорције. Ово може довести до тога да део контејнера буде празан.
- `CenterImage` – исто као и `normal` својство осим што се центар слике поравнава са центром контејнера.
- `AutoSize` - контејнер слике ће се повећати тако да увек може да прикаже целу слику.

Поставити вредност својства `SizeMode` на `StretchImage`. После ове промене, биће приказана цела слика. На слици 9 је слика аутомобила, а црвеном бојом је означен део слике који је био приказан на слици 8 када је вредност својства `SizeMode` било `Normal`.

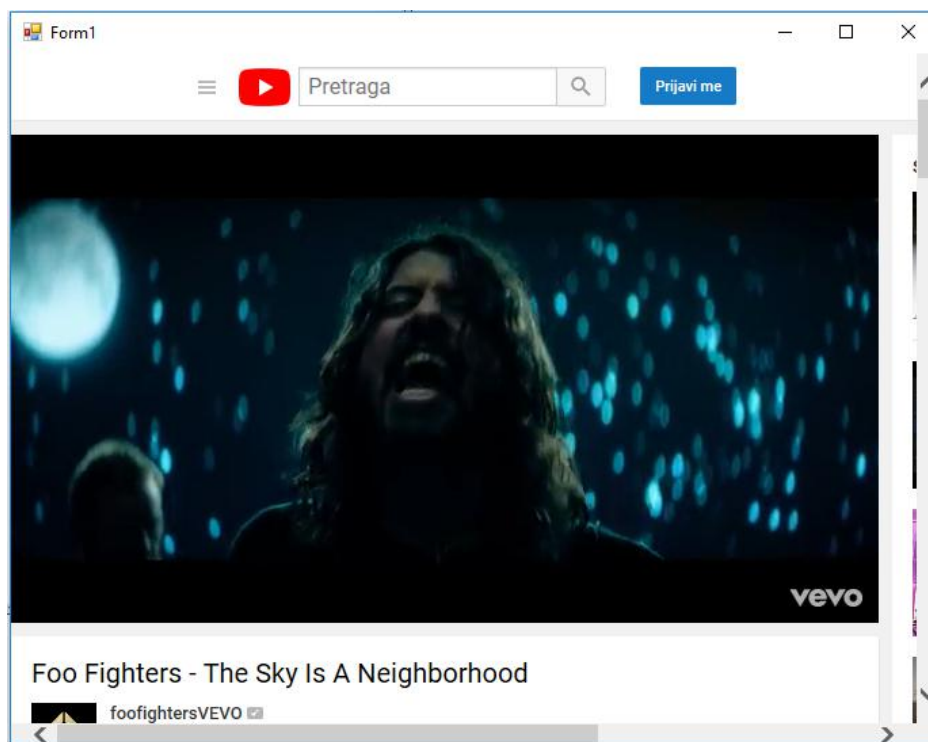


Слика 9. Приказ слике када је вредност својства SizeMode промењена на „StretchImage“

Web browser

Следећа компонента која се налази већ у палети са алаткама је „Web browser“ која уграђује веб клијента (Internet Explorer који је инсталиран на корисничковој машини) и може да прикаже било који садржај. Постављањем ове компоненте на форму, browser ће заузети целу форму. Својство које мења начин приказа Web browser компоненте је „Dock“. Подразумевана вредност је „Fill“, али уколико је потребно дефинисати неке друге димензије из листе опција треба изабрати опцију „None“.

Параметар URL служи за додавање адресе неког сајта који је на Интернету или локалне странице на рачунару. Ко пример биће коришћен је Youtube сајт (слика 10).



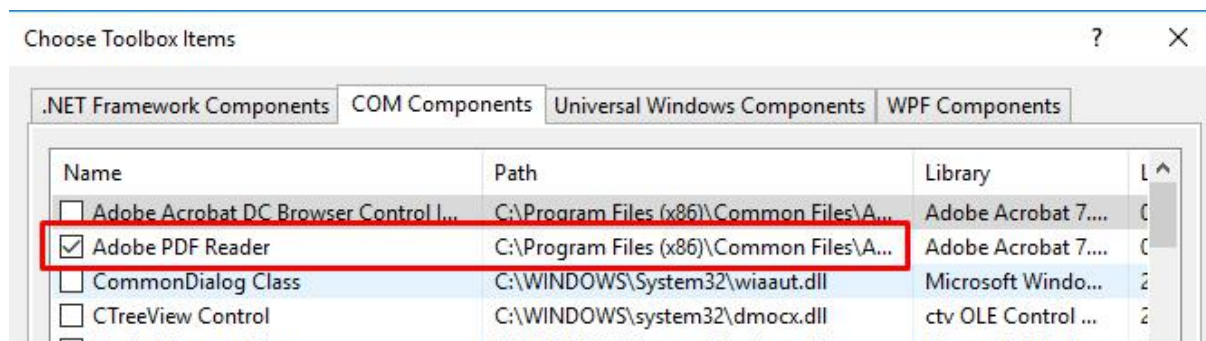


Слика 10. Embedded youtube видео у Windows Forms апликацији

Приликом коришћења ове компоненте потребно је бити свестан њених недостатака. Како компонента користи Internet Explorer као browser за приказивање садржаја, у зависности од верзије инсталиране на рачунару корисника приказ садржаја на страници може да се разликује (верзија 8 која долази подразумевано са Windows 7 и верзија 11 која долази са Windows 10 драстично различито приказују HTML садржај). Не подржава све функционалности HTML језика, као што уграђени садржај на страницама или Flash апликације. Корисник не може из саме Windows форме да контролише садржај на страници.

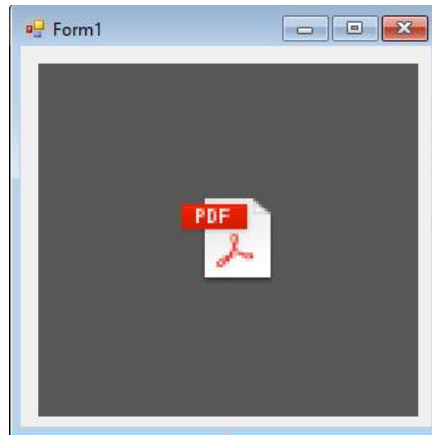
Adobe PDF Reader

PDF документи се често користе. Било коју врсту документа довољно је преbacити у PDF и тако дозволити кориснику да може да чита садржај без могућности да директно мења оригинални документ. Једна од компоненти која може бити коришћена у апликацији, уколико рачунар има инсталиран Adobe Acrobat је Adobe PDF Reader. Овај објекат омогућава приказ PDF датотеке у оквиру апликације. Да би користили овај објекат потребно га је изабарати из „COM components“ на исти начин на који је изабран Windows Media Player, само што ће за убацивање ове компоненте бити потребно да се означи компонента „Adobe PDF Reader“ (слика 11).



Слика 11. Додавање компоненте Adobe PDF Reader

По привлачењу компоненте приказује се оквир са PDF логом (слика 12). Кликом на компоненту, од понуђених својстава треба изабрати URL својство, или Path или неко од својстава са којим се може учитати датотека.

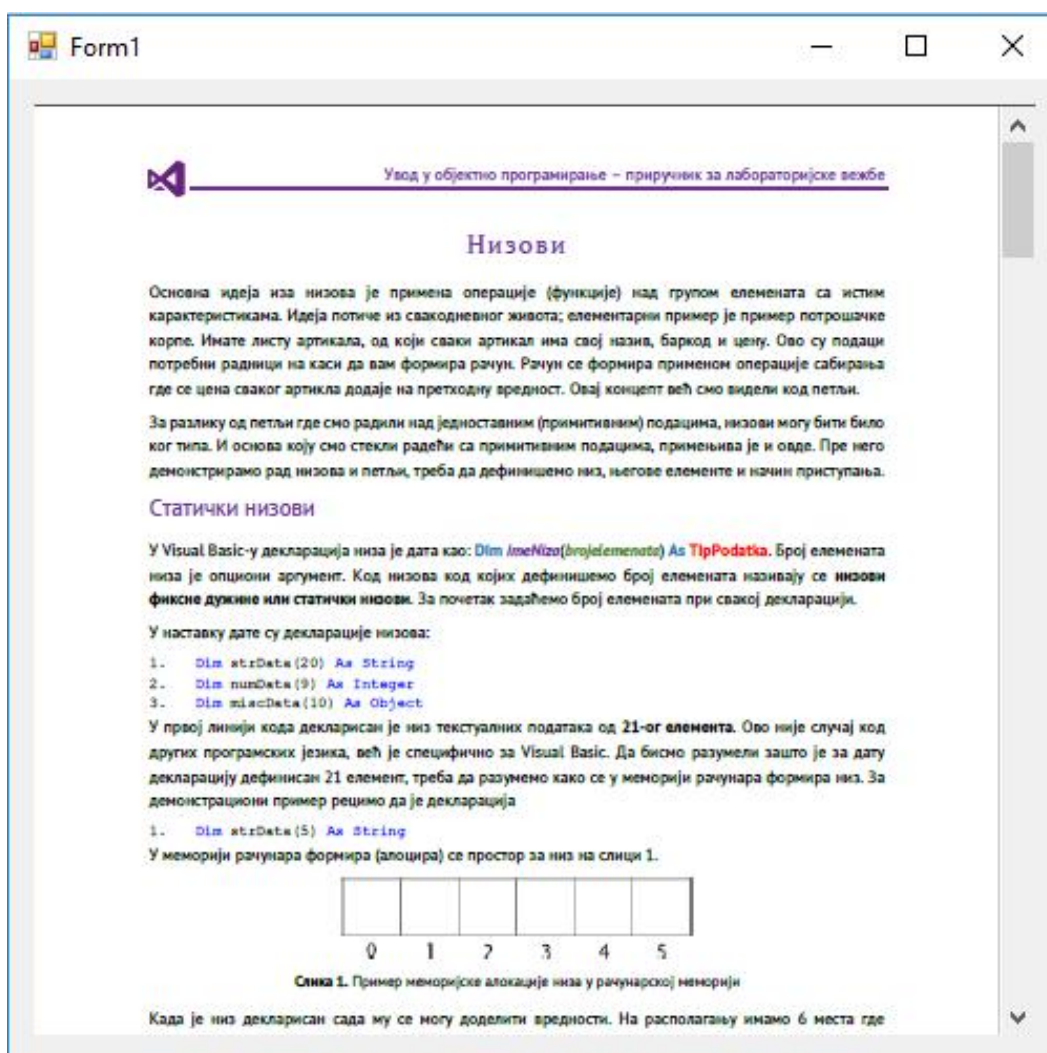


Слика 12. Adobe PDF Reader компонента

Постоји својство „src“ које је обично скраћеница за „source“ и које би могло бити употребљено, али у овој верзији visual Studio-а оно не функционише на прави начин. Следи код за методу Load форме у којој је позван догађај „LoadFile“ у чијем позиву се налази путања до датотеке Nizovo.pdf.

```
480. Private Sub Form1_Load(sender As Object, e As EventArgs) Handles  
    MyBase.Load  
481.     AxAcroPDF1.LoadFile("M:\odrive\Google Suite  
        Drive\Nastava\UOP\Vežba - Nizovi.pdf")  
482. End Sub
```

Приказ документа је дат на слици 13.



Слика 13. Приказ pdf датотеке у оквиру Windows Forms апликације

Недостатак ове компоненте је што корисник мора да има инсталиран Adobe Acrobat. За други тип прегледача PDF датотека морала би да се учита друга, одговарајућа компонента.

Задаци за самостални рад

Апликација за бенд

Креирати апликацију за бенд која када се покрене треба да прикаже промотивни банер (слику) и видео датотеку која пушта видео снимак бенда. Корисник кроз навигацију може да види биографију бенда (текст о бенду и слику), чланове бенда и видео снимке (сваки видео снимак је подмени опција).



Видео демонстрација примера дата је у мултимедијалним материјалима „MME_ThriceApp.mp4“.



БЕЛЕШКЕ





Литература

- [1] Bryan Newsome, *Beginning Visual Basic 2015*, Wrox Publishing 2015, ISBN: 978-1-119-09211-7
- [2] James Foxall, *Visual Basic 2015 in 24 Hours, Sams Teach Yourself*, Sams Publishing 2015, ISBN: 978-0672337451
- [3] David I. Schneider, *Introduction to Programming Using Visual Basic*, Pearson 2017, ISBN: 978-0134542782
- [4] Microsoft Official Visual Basic .NET Guide: <https://docs.microsoft.com/en-us/dotnet/visual-basic/>