VISOKA ŠKOLA ELEKTROTEHNIKE I RAČUNARSTVA STRUKOVNIH STUDIJA

Dr Perica S. Štrbac, dipl. ing.

OBJEKTNO PROGRAMIRANJE 2

Prvo izdanje

Beograd, 2019.

OBJEKTNO PROGRAMIRANJE 2

Autor: dr Perica S. Štrbac, dipl. ing.

Recenzenti: dr Miroslav Lutovac, dipl. ing.
mr Miloš Pejanović, dipl. ing.

Izdavač: Visoka škola elektrotehnike i računarstva strukovnih studija, Beograd, Vojvode Stepe 283, www.viser.edu.rs

Za izdavača: dr Vera Petrović, dipl. ing.

Tehnička obrada: dr Perica Štrbac, dipl. ing.

Dizajn korica: dr Perica Štrbac, dipl. ing.

Godina izdanja: 2019.

Tiraž: 30 primeraka

Štampa: Razvojno-istraživački centar grafičkog inženjerstva TMF, Beograd.

ISBN 978-86-7982-309-0

Odlukom Nastavno-stručnog veća Visoke škole elektrotehnike i računarstva strukovnih studija iz Beograda, na sednici održanoj dana 21.02.2019. godine, odobreno je izdavanje i primena ovog udžbenika u nastavi.

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд

004.42.045(075.8)
004.438JAVA(075.8)

ШТРБАЦ, Перица С., 1968-
Objektno programiranje 2 / Perica S. Štrbac. - 1. izd. - Beograd : Visoka škola elektrotehnike i računarstva strukovnih studija, 2019 (Beograd : Razvojno-istraživački centar grafičkog inženjerstva TMF). - 298 str. : ilustr. ; 25 cm

Tiraž 30. - Bibliografija: str. 287-288. - Registar.

ISBN 978-86-7982-309-0

a) Објектно оријентисано програмирање b) Програмски језик "Java"

COBISS.SR-ID 277463052

POSVETA

Mariji, Pavlu, Jeleni, Jovani, Milici.

Predgovor

Knjiga je napisana prvenstveno studentima treće godine Visoke škole elektrotehnike i računarstva strukovnih studija u Beogradu za predmet Objektno programiranje 2 koji obrađuje objektno orijentisano programiranje u Java tehnologiji.

Knjiga kreće od samog početka objektno orijentisanog programiranja u Java programskom jeziku tako da je podesna i za one koje nisu ranije slušali predmet Objektno programiranje 1 (C++) ili sličan predmet.

Uvodno poglavlje daje smernice za Java tehnologiju i preporuke za integrisano razvojno okruženje.

Poglavlje Java obrađuje programski jezik Java kroz: tipove podataka, kontrole toka programa, operatore, objektno orijentisani pristup, inicijalizaciju statičkih članova i blokova, korišćenje paketa, odabrane klase iz standardne biblioteke te rad sa ulazom i izlazom. Nazivi identifikatora u programskom kodu su većinom dati na engleskom jeziku i intuitivni su čime se lakše pamte nazivi klasa i interfejsa.

U poglavlju Konkurentno programiranje pokazano je kako se kreiraju, kontrolišu i sinhronizuju paraleleni procesi u Java tehnologiji.

Poglavlje koje se odnosi na generičke tipove obrađuje generičke klase, generičke metode, generičke kolekcije, generičke statičke metode, korisničke generičke kolekcije, kao i generičke tipove koji se koriste pri nasleđivanju.

Grafički korisnički interfejs kao poglavlje obrađuje tri tehnologije: AWT (Abstract Windowing Toolkit), Swing i JavaFX. Obrađeni su centralizovana obrada digađaja, obrada pomoću osluškivača, korišćenje lambda izraza te često korišćeni vidžeti .

Klijent-server komunikacija data je u poglavlju koje demonstrira korišćenje Java tehnologije za mrežnu komunikaciju korišćenjem TCP (Transfer Control Protocol) paketa i datagrama u klijent-server višenitnoj arhitekturi kao i URL (Uniform Resource Location) kodovanje i komunikaciju.

Dobro programiranje se može naučiti jedino pisanjem programa i njihovim pokretanjem u datom razvojnom okruženju, čime student stiče znanje, veštinu i iskustvo u pisanju koda, korišćenju razvojnog okruženja te nalaženju i otklanjanju bagova. U ovom smislu knjiga je prožeta velikim brojem primera koji ilustruju opisanu materiju.

Zahvaljujem se svima koji su na bilo koji način doprineli da ova knjiga bude pred Vama, a posebnu zahvalnost dugujem kolegama prof. dr Miroslavu Lutovcu, dipl. ing. i mr. Milošu Pejanoviću, dipl. ing, za korisne sugestije koje su uticale na pisanje ove knjige.

U ovom izdanju udžbenika mogući su propusti, tako da su dobrodošle sve dobronamerne sugestije u cilju poboljšanja sledećeg izdanja.

Knjiga je podržana projektom Ministarstva prosvete, nauke i tehnološkog razvoja Republike Srbije MGKOP – Modernizacija kurikuluma grupe predmeta Objektno programiranje 1 i Objektno programiranje 2 na studijskom programu Računarska tehnika.

U Beogradu, januara 2019.

Autor

Sadržaj

1. UVODNO POGLAVLJE

U knjizi je obrađeno gradivo koje se sluša na trećoj godini osnovnih strukovnih studija smera Računarska tehnika u okviru predmeta Objektno programiranje 2 na Visokoj školi elektrotehnike i računarstva u Beogradu. Tehnologija koja se koristi u knjizi je Java. U trenutku pisanja ove knjige primena programskog jezika Java je premašila milijarde uređaja koji je koriste (*prim. aut.* podatak se dobija pri instalaciji *JDK – Java Development Kit*).

Java je potpuno objektno orijentisani programski jezik tako da je namera autora da čitalac potpuno pređe na OOP način razmišljanja kako bi se na IT tržištu lako mogao uključiti u timove koji razvijaju velike programe (stotine hiljada linija koda).

Čitalac se kroz programske primere postepeno vodi kroz sva poglavlja knjige da bi praktično ovladao izloženom materijom. Autor je pokušao da gradivo predoči tako da se u datom poglavlju koriste znanja iz prethodnih poglavlja. Preporuka je da se primeri prouče, samostalno urade u razvojnom okruženju (preporučujem *JetBrains IntelliJ IDEA* razvojno okruženje), startuju i provere dobijeni izlazi sa izlazima prikazanim u knjizi. Na ovaj način čitalac će ovladati ne samo znanjima već i veštinom u korišćenju odabranog integrisanog razvojnog okruženja koja je neophodna za dobijanje posla na IT tržištu.

Knjiga je pisana u nameri da studenta vodi od nivoa početnika do nivoa izrade konkretnih programa tako da bi u konačnom ishodu student bio u stanju da samostalno rešava zadatke na objektno orijentisan način koristeći Java tehnologiju. Ovi zadaci uključuju mešavinu tehnologija prikazanih u ovoj knjizi (npr. klijent-server višenitno bazirana arhitektura koja za komunikaciju koristi TCP paketnu komunikaciju ili npr. datagrame.

2. JAVA

Cilj poglavlja je da student ovlada: tipovima podataka, kontrolom toka programa, objektno orijentisanim pristupom, inicijalizacijom statičkih članova, korišćenjem paketa, koncepcijom modula, klasama iz biblioteke java.util, radom sa ulazom i izlazom.

Programski jezik Java obuhvata: specifikaciju Java virtuelne mašine (JVM) i specifikaciju programskog jezika Java. JVM se odnosi na platformu baziranu na virtuelnom (izmišljenom) procesoru. Izvorni kod (source code) se prevodi u takozvani Java byte-kod za ovaj virtuelni procesor. Ovim je postignuto da byte-kod Java programa bude identičan na svim platformama. Ovakav kod se ne pokreće direktno već je potreban interpreter koji ovaj kod prilagodi konkretnoj računarskoj platformi za njegovo izvršavanje.

Sun Microsystems, nadalje Sun, kompanija koja je vlasnik programskog jezika Java, nudi besplatan set funkcija za razvoj softvera objedinjen u Java Development Kit (JDK). JDK obuhvata JVM interpreter, kompajler i skup razvojnih alata koji se pokreću iz prompta (komandne linije). Sun je izdao JDK paket za nekoliko računarskih platformi: Windows, Linux/Intel, Solaris/SPARC i Solaris/Intel.

Java specifikacija je javno dostupna što omogućuje da drugi proizvođači kreiraju svoje implementacije Jave za datu računarsku platformu.

Postoje i softveri koji mogu generisati iz Java izvornog koda izvršni kod za datu računarsku platformu (npr. TowerJ generiše izvršni kod za Windows).

Java programski jezik korene vuče iz jezika C. Java je potpuno objektno orijentisan programski jezik za razliku od C++ koji je hibridni jezik. U ovom poglavlju ćemo se baviti specifikacijom programskog jezika Java. Sledi popis značajnih karakteristika Jave:

- radi se o objektno orijentisanom jeziku opšte namene;
- prenosivost programa napisanog u Javi je zagarantovana specifikacijom JVM interpretera;
- veoma malo zavisi od karakteristika konkretnog računarskog sistema na kome se izvršava;
- brzina izvršavanja Jave je smanjena jer se koristi interpreter, ali vremenom taj nedostatak je zbog korišćenja sve bržih hardverskih platformi sve manji;
- pogodna je za pisanje konkurentnih (paralelnih), mrežnih i distribuiranih programa.

Referentna dokumentacija za Javu data je sajtu firme JavaSoft, koja je u okviru Sun-a (http://java.sun.com). Na sajtu http://www.bruceeckel.com data je besplatna elektronska verzija knjige Thinking in Java (autor Bruce Eckel).

2.1. Tipovi podataka

U programskom jeziku Java koriste se ugrađeni tipovi i korisnički tipovi podataka. Ugrađeni tipovi podataka pored osnovne funkcije služe i za izgradnju složenijih struktura podataka unutar korisničkih tipova podataka. Ugrađeni tipovi podataka obuhvataju osnovne (primitivne) tipove i jedan broj izvedenih tipova (nizovi, funkcije, reference, konstante, nabrajanja). Primitivni tipovi dati su u tabeli 2.1. Ovi tipovi podataka su model po kome se elementarni podaci predstavljaju na datoj mašini. Naziv identifikatora u Javi 9 ne može više biti _ (donja crta).

Tabela 2.1. Primitivni tipovi podataka

Primitivni tip	Veličina u bitovima	Minimum	Maksimum
boolean	1	-	-
char	16	Unicode 0	Unicode $2^{16}-1$
byte	8	-128	+127
short	16	-2^{15}	$+2^{15}-1$
int	32	-2^{31}	$+2^{31}-1$
long	64	-2^{63}	$+2^{63}-1$
float	32	±1.18e-038 IEEE754	±3,4e+038 IEEE754
double	64	±2,23e-308 IEEE754	±1,8e+308 IEEE754
void	-	-	-

Poznavaoci programskog jezika C++ bi trebalo da obrate pažnju na tip char koji u Javi zauzima dva bajta da bi se predstavili karakteri Unicode kodnog standarda. Ovaj kodni standard obuhvata karaktere svih današnjih jezika.

Korisnički tipovi podataka su klase. Ove tipove podataka kreira korisnik prema pravilima objektno orijentisanog programiranja.

2.2. Kontrola toka programa

Kontrolu toka programa određuju standardne konstrukcije preuzete iz jezika C++:

- uslovne naredbe,
  ako je uslov `uslov1` zadovoljen izvršava se blok naredbi `naredba/e`:

  ```
  if (uslov1) {naredba/e}
  ```

  ako je uslov `uslov2` zadovoljen izvršava je blok naredbi `naredbe2` inače se izvršava blok naredbi `naredbe3`:

  ```
  if(uslov2){naredbe2}
  else      {naredbe3}
  ```

  ako je vrednost izraza `izraz1` u `switch`-u jednaka nekoj vrednosti iza ključne reči `case` izvršiće se blok naredbi koji sledi dati `case`, a u slučaju da niti jedna `case` vrednost ne odgovara izrazu `izraz1` u `switch`-u, izvršiće se blok naredbi iza ključne reči `default` (`naredbe6`):

  ```
  switch (izraz1) {
        case vrednost1: naredbe4; break;
        case vrednost2: naredbe5; break;
        deafult: naredbe6;
  }
  ```

- iterativne,
  programska petlja pre koje se izvrši inicijalizacija `inic` i koja izvršava blok naredbi `naredbe8` sve dok je zadovoljen uslov `uslov3`, pri čemu se na kraju svake iteracije izvrši `naredbe7,` a onda ponovo ispituje `uslov3` za novu iteraciju (ako se ne navede, podrazumevano je `uslov3` zadovoljen):

  ```
  for(inic; uslov3; naredbe7){naredbe8}
  ```

  programska petlja koja izvršava blok naredbi `naredbe9` sve dok je zadovoljen uslov `uslov4`:

  ```
  while(uslov4) {naredbe9}
  ```

  programska petlja koja jednom izvršava blok naredbi `naredbe10,` a onda proverava da li je zadovoljen uslov `uslov5` za novu iteraciju:

  ```
  do {naredbe10} while(uslov5);
  ```

  – naredbe skoka,
  ako se želi izlazak iz prvog okružujućeg bloka (izlazak iz petlje):

break;

ako se želi skok na kraj tekuće iteracije:

continue;

Za lakšu čitljivost programa služe komentari. Pisanje komentara je kao u C++-u:

```
// odavde pa nadalje sledi komentar u ovom redu
/* početak komentara u ovom redu
nastavak komentara
završetak komentara u ovom redu do znaka */
```

2.3. Operatori

Operatori u Javi su gotovo isti kao i u C++. Razlikuju se sledeće grupe operatora:

- aritmetički operatori:

`+ - * /`	sabiranje, oduzimanje, množenje, deljenje
`%`	ostatak pri celobrojnom deljenju
`+= -= *= /=`	operacija sabiranja, oduzimanja, množenja i deljenja uz dodelu, npr. `a+=5;` je isto što i `a=a+5;`

`++ --`	autoinkrement, autodekrement, npr.
	`a++;` je isto što i `a=a+1;`
	`a--;` je isto što i `a=a-1;`

- relacioni operatori:

`==`	jednako
`<`	manje
`>`	veće
`!=`	različito
`>=`	veće ili jednako
`<=`	manje ili jednako

- logički operatori (&&, ||, !);

`&&`	logičko i

`||`	logičko ili
!	logička negacija

- bit-operatori (*bitwise* operatori) koji operišu na nivou bita:

`&`	operator i
`|`	ili
`~`	komplement
`^`	ekskluzivno ili
`>>`	aritmetičko pomeranje u desno
`>>>`	logičko pomeranje u desno
`<<`	pomeranje u levo
`&= |= ~= ^=` `>>= >>>= <<=`	kao prethodno samo uz dodelu, npr. `a&=5;` je isto što i `a=a&5;`

- operator dodele (=); npr. a = b;
- trojni operator uslovne dodele (`?:`). npr.
  `c = (a<b)? true: false;`
  ispituje se da li je `a<b`, ako jeste `c` postaje true inače postaje false.
  `d = (a<b)? a: b;`
  ispituje se da li je `a<b`, ako jeste `d` dobija (dodeljuje se) vrednost a inače dobija vrednost b.

Treba paziti na to da je boolean tip strogo logički tako da se ne može koristiti kao u C++, na primer za petlju koja se vrti beskonačno gde je stalno zadovoljen uslov za iteraciju. U C++ može se pisati `while(1)`, ali u Javi izraz mora biti strogo logičkog tipa, u ovom slučaju poput `while(true)` ili `while(2>1).`

2.4. Objektno orijentisani pristup

Objektno orijentisano programiranje (OOP) je nastalo kao rešenje za softversku krizu koja počinje krajem sedamdesetih godina sa porastom softverskih zahteva. Osnovni problem je bio kako efikasno održavati ogromne softverske programe. Objektno orijentisano programiranje je poseban način razmišljanja pri projektovanju programa i za razliku od algoritamskog (tradicionalnog) programiranja, gde se razmišlja o rešenju problema u koracima, kod objektnog pristupa se razmišlja o objektima koji čine problem i njihovoj interakciji.

Objekti su opisani stanjem i ponašanjem. Oni međusobno komuniciraju i time mogu menjati svoja stanja. Pojmom klase se opisuje ponašanje objekta i

njegovih stanja. Stanja i ponašanja predstavljaju članove date klase. Stanja objekata su opisana podacima članovima (atributima), a ponašanja objekta su opisana funkcijama članicama (metodama).

Konkretan primerak klase zove se objekat neke klase i za svaki kreirani objekat kontruiše se njegovo početno stanje. Jednom napisan OOP program može se lako modifikovati i ponovo upotrebiti (software reuse).

Ključni koncepti OOP-a su: klase, objekti, apstrakcija, enkapsulacija, nasleđivanje i polimorfizam.

2.4.1. Klase i objekti

Sledi primer programa za upoznavanje Jave koji ispisuje "Hello world!" (slika 2.1). Data je definicija klase Hello koja počinje ključnom rečju **`class`** iza koje sledi identifikator Hello, otvorena velika zagrada, metoda main i zatvorena velika zagrada iza koje ne sledi tačka-zarez za razliku od C++-a.

```
class Hello {
    public static void main(String args[]) {
      System.out.println("Hello world!");
  }
}
```

Slika 2.1. Klasa Hello

Java je potpuno objektno orijentisan programski jezik, tako da je i metoda `main` (od koje počinje izvršavanje programa) u okviru klase (za razliku od C++-a). Svaka klasa može imati metodu `main`, a program će početi da se izvršava od one main metode čija je klasa navedena u pozivu za izvršavanje kompajliranog programa. Za datu klasu se main metod može iskoristiti bar za potrebe testiranja same klase.

Reč **`public`** označava javno pravo pristupa čime je omogućen poziv `main` metode van klase u kojoj se nalazi, u našem slučaju to je poziv iz komandne linije.

Reč **`static`** određuje da je metoda na nivou cele klase, odnosno, da može da se pozove bez kreiranja konkretnog objekta te klase.

Reč **`void`** govori da se metoda main ponaša kao procedura, odnosno, ne vraća vrednost (vraća **`void`**). Formalni argument args predstavlja niz String-ova koji se popunjava parametrima koji se navode u komand promptu pri pozivu main-a ove klase (java nazivklase parametri).

Linija koda `System.out.println("Hello world!");` ispisuje na konzoli tekst `Hello world!`. Ovde se radi o pozivu statične metode println koja pripada klasi

out koja kao statički član pripada klasi System. Nakon naredbe sledi tačka-zarez kao graničnik naredbe (kao u C++).

Prethodni kod mora biti smešten u datoteku koja ima naziv koji odgovara nazivu klase koju definiše i čija ekstenzija ima naziv java (u ovom slučaju to je datoteka Hello.java). Java razlikuje velika i mala slova (case-sensitive) kao i C++ tako da i ovaj uslov mora biti zadovoljen.

Nakon instaliranja JDK i pisanja prethodnog koda u tekst editoru ili editoru razvojnog okruženja prevođenje ovog izvornog koda u byte-kod se obavlja pozivom iz komandne linije:

```
javac Hello.java
```

Rezultat će biti byte-kod koji ima isti naziv kao i naziv klase koja se prevodi i ekstenziju čiji je naziv class (u ovom primeru rezultat je `Hello.class`). Naziv programa javac potiče od Java-compiler. Pokretanje programa `Hello.class` obavlja se pozivom iz komandne linije:

```
java Hello
```

pri čemu se ne navodi ekstenzija class. Ova naredba će pozvati metodu main klase Hello. Ako navedena klasa nema main metodu dojavila bi se greška.

Podrazumeva se da se pre kucanja ovih dveju komandi navigacija u komand promptu podesi na folder koji sadrži kod Hello.java (pri instalaciji JDK potvrditi postavljanje promenljivih PATH i CLASSPATH).

Apstrakcija tipova podataka (abstract data types) omogućuje da korisnik (programer) proizvoljno definiše svoje tipove i potpuno ravnopravno ih koristiti (Fen, Printer, Jabuka, Automobil, Bankovniracun, Klijent, ...).

Neka je dat sledeći programski kod koji definiše klasu koja opisuje fen za sušenje kose (slika 2.2). Fen je spojen svojim kablom na utičnicu koja ima uredno napajanje i inicijalno je on isključen (dugme off) tako da ne duva:

```
class Fen{

  boolean bUkljucen;

  Fen(){
    bUkljucen = false;
  }

  void ukljuci() {
    bUkljucen = true;
  }

  void iskljuci() {
```

```
    bUkljucen = false;
  }

  public static void main(String []args){
    Fen fen_1 = new Fen();
    fen_1.ukljuci();
    Fen fen_2 = new Fen();
  }
}
```

Slika 2.2. Klasa Fen

Ideja je da se napiše opšti opis nekog elementa, ovde je to fen za kosu koji ima svoje stanje i ponašanje. Ovaj opšti opis za sve elemente kojima su zajedničke osobine stanja i ponašanja se naziva klasom.

Konkretan primerak klase predstavlja objekat te klase. Svaki kreirani konkretni element (objekat ili instanca klase) imaće svoje stanje i svoje ponašanje. Primer su dva fena gde se jedan fen uključi, a drugi bude ostavljen u inicijalnom (podrazumevanom) stanju, odnosno, isključen.

Preporuka je da naziv klase bude ispisan velikim početnim slovom (`Fen`). U konkretnom kodu atribut koji određuje stanje objekta je logički podatak član `bUkljucen`. Prefiks b je stavljen da bi asocirao na boolean tip podatka, što je korisno za čitljivost programa. Stanja objekta klase Fen mogu biti: `bUkljucen` je `true` (duva) ili `false` (isključen).

Ponašanje objekta klase Fen određeno je metodama `ukljuci` i `iskljuci`. Pozivom prve metode postavićemo stanje objekta takvo da je taj fen uključen, dok pozivom druge metode biće postavljeno stanje objekta da je taj fen isključen. Obe ove metode nemaju formalne argumente (ne prosleđuju se parametri ovim metodama), a metode imaju povratnu vrednost `void` što znači da se pozivanjem bilo koje od ovih metoda one koriste kao procedure, a ne kao funkcije.

Posebna metoda koja ima isti naziv kao i naziv klase zove se konstruktor te klase. Konstruktor klase nema povratnu vrednost, čak ni `void`. Osnovna namena konstruktora je da kreira objekat i postavi njegovo početno stanje inicijalizujući podatke članove.

U navedenom primeru inicijalno stanje će biti da je fen isključen (podatak član `bUkljucen` dobija vrednost `false`). Ako je podatak član takođe objekat, onda će konstruktor pozivom odgovarajućeg konstruktora ovog objekta izvršiti njegovo kreiranje i inicijalizaciju.

Povratna vrednost **`void`** ima isto značenje kao i u metodama `ukljuci` i `iskljuci`. Argument `args` predstavlja niz objekata tipa String i koristi se u

metodi `main` za prihvatanje parametara iz komandne linije (prim. aut. navedeno u ovom pasusu biće kasnije detaljno obrađeno). Tri linije koda u metodi main znače respektivno kao što sledi:

```
Fen fen_1 = new Fen();
```

- promenljiva čiji je identifikator reč `fen_1` je referenca (kao pokazivač u C++ bez pointerske aritmetike i dodeljivanja proizvoljnih vrednosti) koja se odnosi na kreirani dinamički objekat klase `Fen`. Kreiranje dinamičkog objekta se postiže korišćenjem ključne reči new iza koje sledi poziv konstruktora date klase. U ovom slučaju konstruktor klase `Fen` konstruisaće objekat u kom će podatak član ovog objekta `bUkljucen` postaviti na neistinitu (false) vrednost. Sada je `fen_1` kreiran i u isključenom je stanju.

```
fen_1.ukljuci();
```

- ovom linijom koda pristupa se metodi `ukljuci` objekta `fen_1`. Pozvana metoda `uključi` postaviće podatak član `bUkljucen` objekta `fen_1` na istinitu (true) vrednost. Sada je `fen_1` uključen.

```
Fen fen_2 = new Fen();
```

- kreiranje reference na dinamički objekat klase `Fen` čiji je identifikator reč `fen_2` i postavljanje njegovog podatka člana `bUkljucen` na neistinitu (false) vrednost.

Prevođenje i pokretanje programa `Fen` obavlja se sledećim komandama iz komandne linije:

```
javac Fen.java
java  Fen
```

Postoji nekoliko razlika između C++-a i Jave koje su bitne pri prelasku sa prvog na drugi jezik jer kod Jave:

- je podrazumevano pravo pristupa javno, tako da je podatak član `bUkljucen` javni. Ovo znači da mu se može pristupiti van klase.
- nije moguće definisati promenljive i funkcije izvan klase (ne mogu se definisati ni globalne promenljive niti globalne funkcije)
- postoji samo definicija klase (nema odvojene deklaracije i definicije klase).

Posmatra se primer klase `ProbajFen`. U napred navedenom primeru u okviru klase `Fen` bila je i metoda `main`, a sada se proba fenova (klase `Fen`) obavlja iz `main` metode koja je napisana u okviru klase `ProbajFen` (slika 2.3).

```
class ProbajFen{
```

```
  public static void main(String []args){
    Fen fen;
    fen = new Fen();
    fen.ukljuci();
    fen.bUkljucen = false;
  }
}
```

Slika 2.3. Klasa ProbajFen

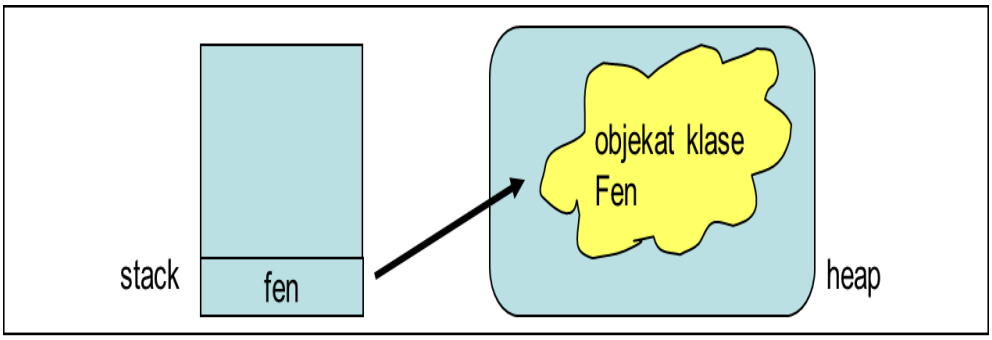
Postupak pokretanja programa `ProbajFen` ide analogno prethodnim primerima:

```
javac ProbajFen.java
java  ProbajFen
```

Pozivom `javac ProbajFen.java` automatski će se pozvati kompilacije svih klasa koje se pominju u datoteci `ProbajFen.java` (ovde je to klasa `Fen`) kako bi se formirala datoteka `ProbajFen.class` jer program mora da zna i byte-kod klase `Fen`.

Pozivom `java ProbajFen` izvršava se `main` metoda u okviru klase `ProbajFen` tako da će se prvo kreirati identifikator `fen` koji predstavlja referencu na objekat klase `Fen`.

U drugoj liniji koda referenca `fen` sada pokazuje na novokreirani objekat klase `Fen`. Sledeća linija koda preko reference `fen` poziva metodu `ukljuci` ovog konkretnog objekta klase `Fen`. Poslednja linija koda preko reference direktno pristupa podatku članu objekta `bUkljucen` koji ima javno (podrazumevano) pravo pristupa i postavalja mu vrednost `false`.



Slika 2.4. Referenca fen ukazuje na objekat klase Fen

Na slici 2.4. dat je prikaz stanja steka (*stack*) i hipa (*heap*). Promenljiva `fen` je lokalna promenljiva u metodi `main` i smešta se na stack, dok se memorija za objekat klase `Fen` zauzima na heap-u programa.

Referenca može imati ili vrednost adrese inicijalizovanog objekta klase za koju je definisana ili `null` vrednost u slučaju kada ne ukazuje na konkretni objekat date klase (nula-pokazivač u C++) ili da joj se dodeli referenca na objekat iste klase.

Posmatra se kreiranje dva objekta klase *Fen* i inicijalizacija dve reference koje respektivno ukazuju na ova dva objekta.

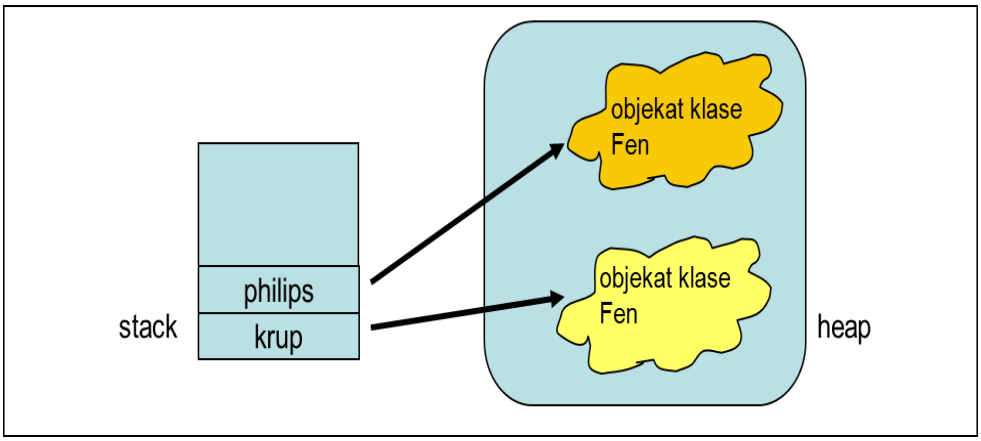
```
Fen krup    = new Fen();
Fen philips = new Fen();
```

Reference se nalaze na stack-u programa, dok su objekti smešteni na heap-u.

Prvo se na stack postavlja referenca `krup` i nakon kreiranja na heap-u prvog objekta klase `Fen` referenca se inicjalizuje tako da ukazuje na ovaj objekat.

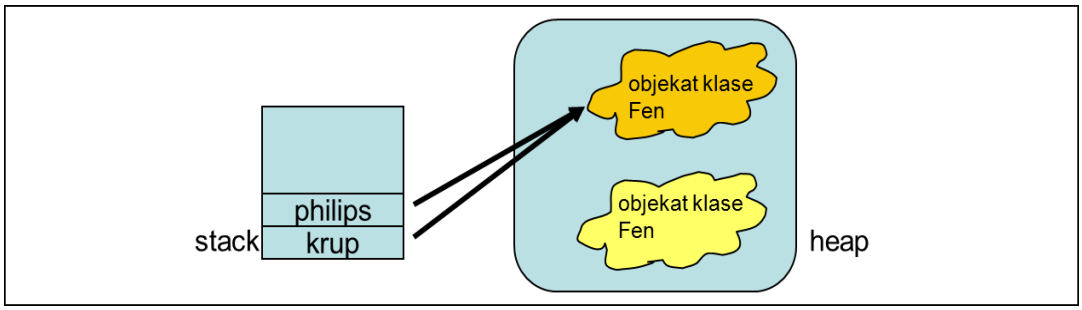
Nakon ovoga na stack se postavlja referenca `philips` koja se inicijalizuje da ukazuje na drugi kreirani objekat klase Fen koji je smešten na heap-u.

Na slici 2.5. grafički je prikazano stanje stack-a i heap-a nakon ove dve naredbe.



Slika 2.5. Dve refence koje ukazuju na dva objekta

Programski jezik Java nema naredbu za dealokaciju memorije, kao što je u C++ to operator delete te tu ulogu preuzima poseban proces (demonska nit) koji se aktivira paralelno sa glavnom niti (glavnim programom). Ovaj proces se



Slika 2.6. Nedostupnost objekta na koji više ne ukazuje referenca

zove skupljač đubreta (garbage collector GC). Programer nema nikakav uticaj na način rada GC (moguć je poziv `System.gc()`). Uslov da bi ovaj proces mogao da ukloni objekat iz memorije je da ne postoji referenca koja ukazuje na dati objekat, drugim rečima, kada objekat više nije dostupan. Ako se ovakav objekat ne bi uklonio iz memorije došlo bi do curenja memorije (memory leaking).

U Javi 9 postoje 4 garbage collector-a. Podrazumevani GC u verziji 9 je G1 koji se pokazao kao odličan za heap-ove veće od 4 GB, pravi manje frekventne, ali duže pauze.

Neka se sada izvrši sledeća linija koda:

```
krup = philips;
```

sada će objekat na koji je ukazivala referenca `krup` biti nedostupan čime postaje kandidat za garbage collector u smislu dealokacije memorije na heap-u koji je taj objekat zauzimao. Stanje stack-a i heap-a nakon ove linije koda prikazano je na slici 2.6.

Garbage collector samostalno oslobađa memoriju nedostupnih objekata te samostalno defragmentira memoriju. Programer nema uticaja na način rada GC i jedino što mu preostaje da pre uklanjanja preostale reference na objekat oslobodi sve resurse koje je taj objekat zauzimao (mrežne konekcije, datoteke,...).

Java ima specijalnu metodu `finalize` (nema destruktore kao C++) koja se poziva pre nego što GC oslobodi memoriju koju je zauzimao objekat.

Garbage collector sam određuje kada uklanja objekat iz memorije tako da nema garancije da će se to i desiti. Može se desiti da `finalize` metoda ne bude pozvana npr. u slučaju da ima dovoljno radne memorije na raspolaganju i nema potrebe da garbage collector interveniše čime bi resursi koje je koristio objekat ostali zauzeti.

Sledeći primer ispisuje na konzoli rečenicu `"Pozvan je konstruktor!"` i verovatno garbage collector nikada neće pozvati metodu `finalize` (slika 2.7).

```
class ProbaKonstruktoraIFinalize{

   ProbaKonstruktoraIFinalize() {
      System.out.println("Pozvan je konstruktor!");
   }

   protected void finalize() throws Throwable {
      System.out.println("Pozvana je metoda finalize!");
   }

   public static void main(String[] args) {
      ProbaKonstruktoraIFinalize pkf =
        new ProbaKonstruktoraIFinalize ();
      pkf = null;
```

```
        System.out.println("main running...");
    }
}
```

Slika 2.7. Klasa ProbaKonstruktoraIFinalize

U `main` metodi se inicijalizuje referenca `pkf` kreiranim objektom klase `ProbaKonstIFinalize`. Kreiranje ovog objekta radi konstruktor koji je pozvan iza ključne reči `new` pri čemu konstruktor na konzoli ispiše rečenicu `Pozvan je konstruktor!.`

U sledećoj liniji koda referenci `pkf` se dodeljuje `null`, što znači da više ne ukazuje na prethodno kreiran objekat. Sada je objekat kandidat za garbage collector i pitanje je da li će memorija koju je zauzimao objekat biti oslobođena.

Ako se na konzoli ispiše: `"Pozvana je metoda finalize!",` onda je garbage collector oslobodio memoriju koju je objekat zauzimao.

2.4.2. Metode

Metode se u opštem slučaju definišu navođenjem prava pristupa (modifikatora pristupa), posebne oznake, povratnog tipa, identifikatora, liste parametara i tela metode (složenog bloka).

Prava pristupa omogućuju relizaciju enkapsulacije (encapsulation) kojom se postiže skrivanje detalja realizacije nekog tipa. Ovo znači da se korisnicima tipa precizno definiše samo šta se sa tipom može raditi, a način kako se to radi (definisanje) sakriti od korisnika.

Prava pristupa su:

- javno (`public`) po kom je metoda vidljiva iz celog programa i ovo je podrazumevano pravo;
- zaštićeno (`protected`) po kom je metoda vidljiva za objekte klase koja nasleđuje klasu kojoj pripada metoda;
- privatno (`private`) po kom je metoda vidljiva samo u okviru klase kojoj pripada;
- nespecificirano (`friendly`) po kom je metoda vidljiva za klase iz istog paketa.

Navedeno vredi i za podatke članove ispred čije definicije se navodi modifikator pristupa. Modifikator pristupa može se naći ispred definicije klase što znači njegovu primenu na celu klasu.

Pomenuti pojam paket se odnosi na organizovanje klasa. Paketi i klase su organizovani kao folderi i datoteke respektivno. Klasa koja nije u korenskom paketu mora u okviru svoje datoteke imati odgovarajuću deklaraciju koja određuje u kom se paketu nalazi:

```
package mojpaket;
class Fen { ... }
```

Ova deklaracija mora biti prva navedena deklaracija u datoteci u kojoj se definise klasa. Prethodne linije koda znače datoteka `Fen.java` mora biti smeštena u folderu `mojpaket` koji se nalazi u korenskom folderu. Prevođenje i pokretanje bi se obavilo sledećim komandama:

```
...\korenskipaket\>javac mojpaket\Fen.java
...\korenskipaket\>java  mojpaket.Fen
```

Ako se hoće da klasa Fen bude u podfolderu mojpaket2 koji se nalazi u folderu mojpaket, onda bi deklaracija bila:

```
package mojpaket.mojpaket2;
class Fen { ... }
```

dakle, separator naziva paketa je tačka, pa bi adekvatni pozivi bili:

```
...\korenskipaket\>javac mojpaket\mojpaket2\Fen.java
...\korenskipaket\>java  mojpaket.mojpaket2.Fen
```

Posebne oznake ispred metoda su `static` i `final`. Oznaka `static`, kao što je već navedeno, znači da metoda pripada klasi i da se može pozvati navođenjem samo imena klase, operatora pristupa metodi (tačka), naziva metode i liste njenih stvarnih argumenata.

Oznaka `final` govori da se metoda ne može redefinisati (nadjačati) prilikom nasleđivanja date klase.

Tip rezultata metode može biti primitivni tip ili referenca na objekat, osim konstruktora koji nema povratni tip.

Identifikator predstavlja naziv metode.

Lista parametara može biti bez formalnih argumenata ili sa formalnim argumentima koje čine primitivni tipovi i reference na objekte. Primitivni tipovi se kao argumenti prenose po vrednosti, dok za reference na objekte vredi prenos po adresi osim referenci na objekte klase String.

Primer prenosa argumenta tipa String:

```
public void dodajAneks(String strPoruka) {
        strPoruka += "Aneks";
}
```

U ovoj metodi će referenca `strPoruka` biti lokalna referenca koja će ukazivati na novokreirani objekat klase String koji samo po sadržaju odgovara originalu. Sada će se konkatenacija (spajanje) sa stringom `"Aneks"` izvesti na

novokreiranom objektu klase String, što nema uticaja na sadržaj originalnog stringa.

Povratkom iz metode `dodajAneks` ostaje samo originalna referenca na stari objekat klase String. Iako u Javi nema redefinisanja operatora (za razliku od C++) operator + se koristi za konkatenaciju stringova. Ako neki operand u operaciji konkatenacije nije String tada se za primitivne tipove vrši implicitna konverzija u tip String, a ako su u pitanju reference na objekte klasa, onda se konverzija u String dobija implicitnim pozivom metoda toString ovih objekata.

Primer:

```
int broj = 10;
String str = "Broj je : "+broj; //str="Broj je : 10"
BiloKojaKlasa bkk = new BiloKojaKlasa();
str+=bkk;   //isto kao i str=str+bkk.toString();
```

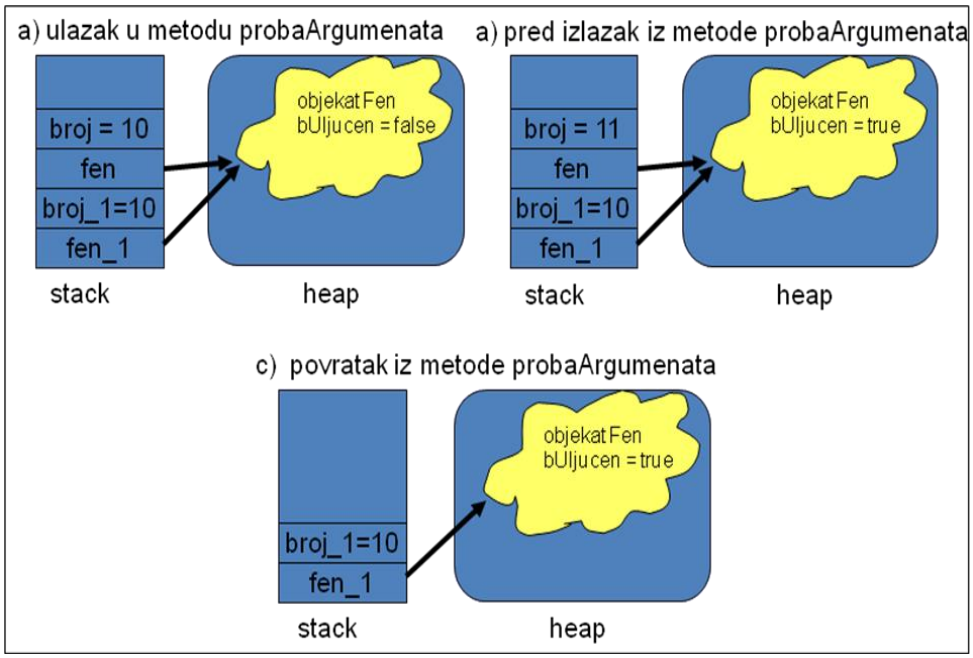
Posmatrajmo sledeću metodu kao primer za prenos stvarnih argumenata po vrednosti i po adresi:

```
public void probaArgumenata(Fen fen, int broj)  {
      fen.ukljuci();
      broj++;
}
```

Izvršavanjem sledećeg koda :

```
Fen fen_1 = new Fen();
fen_1.bUkljucen = false;//primer za direktan pristup
                        /*iako ovo vec uradi
                          konstruktor klase Fen */
int broj_1 = 10;
probaArgumenata(fen_1, broj_1);
```

dešavaće se promene stanja stack-a i heap-a koji odgovara prikazu na slici 2.8.



Slika 2.8. Prenos parametara po vrednosti i po adresi

Prva linija koda će inicijalizovati referencu `fen_1` da ukazuje na upravo-kreirani objekat klase `Fen`. Nakon toga sledi kao primer direktnog pristupa podatku članu postavljanje podatka člana `bUkljucen` objekta `fen_1` na vrednost `false`.

U trećoj liniji koda inicijalizuje se promenljiva tipa int (primitivni tip) brojem 10. Sledi poziv metode `probaArgumenata` kojoj su prosleđena dva stvarna argumenta, prvi je referenca `fen_1`, a drugi je kopija vrednosti promenljive `broj_1`.

Sada formalni argument `fen` kao referenca na objekat klase `Fen` ukazuje gde i referenca `fen_1`, a formalni argument `broj` ima vrednost kao i promenljiva `broj_1`.

Izvršavanjem prve linije koda u pozvanoj metodi poziva se metoda `ukljuci` objekta na koji ukazuje referenca `fen`, a to je isti objekat na koji ukazuje referenca `fen_1` te podatak član `bUkljucen` objekta `fen` dobija istinitu vrednost.

Sledeća linija koda inkrementira vrednost formalnog argumenta `broj` koji nema uticaja na stvarni argument `broj_1` jer je prenos bio po vrednosti.

Metode se razlikuju imenom i listom parametara. Ovim je postignuto da više metoda mogu imati isti identifikator, ali moraju imati različite liste argumenata. Ovakav pristup se naziva preklapanje metoda (*method overloading*). Metode se ne razlikuju po povratnoj vrednosti. Konstruktor kao metoda takođe može biti preklopljen.

Primer klase sa preklopljenim konstruktorima i metodama dat je na slici 2.9.

```
class Preklopljena {
      Preklopljena()           { ... }
      Preklopljena(String str) { ... }
      int prikazi()         { ... }
      double prikazi(int i) { ... }
      int prikazi(String s) { ... }
}
```

Slika 2.9. Primer preklapanja konstruktora i metoda

Rezervisana reč this koristi se kao što sledi:

- Poziv preklopljenog konstruktora što se izvodi sledećim pozivom:
  this(lista argumenata)
  gde lista argumenta odgovara listi argumenata pozvanog preklopljenog konstruktora. Ovaj poziv mora biti prvi u okviru datog konstruktora.
- Referenciranje objekta koji je pozvao ovu metodu (tekući objekat).
- Razrešavanje istog imena i parametra metode i člana klase kome se on dodeljuje, da bi se znalo šta se kome dodeljuje.

Sve nabrojano dato je na primeru klase MyThis (slika 2.10).

```
class MyThis{
   double cena;
   String naziv;
   MyThis(){
      this.cena = 0;
      this.naziv = "";
   }
   MyThis(double cena, String naziv){
      this.cena = cena; // razresavanje imena
      this.naziv = naziv;
   }
   MyThis(String naziv){
      this(0, naziv);   // poziv preklopljenog konstruktora
   }
   void Ispisi(){
      System.out.println("Opis: " + this.naziv +" "+ this.cena);
      // referenciranje tekuceg objekta
   }
}
```

Slika 2.10. Korišćenje rezervisane reči this

Videlo se da se ključna reč `final` može naći ispred naziva metode (sprečava nadjačavanje, override, te metode u izvedenim klasama), ali isto tako se ova ključna reč može naći ispred definicije atributa. Ako se nađe ispred atributa, označava da taj atribut ima konstantnu vrednost koja se ne može menjati pri čemu je inicijalizacija obavezna (moguće je inicijalizaciju postaviti „kasnije" u konstruktoru).

Primer „tradicionalne" konstante :

```
final double PI = 3.14;
```

Ako se ključna reč `final` navede ispred klase, onda ta klasa ne može biti superklasa (ne može se proširiti tj. nasleđivati). Korišćenjem reči final ispred varijable ili parametra učiniće ih nepromenljivim. Ako se koristi ključna reč final ispred reference, onda se referenca ne sme preusmeriti na drugi objekat.

Takođe se i ključna reč `static` može osim ispred naziva metode naći i ispred definicije atributa, a to znači da taj atribut pripada klasi, a ne objektima u smislu da nije potrebno kreirati objekat klase da bi se koristio taj atribut. Ako se kreiraju objekti date klase, svi će deliti istu vrednost statičkog atributa.

Ako se pri deklaraciji navedu ključne reči static final, onda se inicijalizacija radi na mestu deklaracije ili u statičkom inicijalizacionom bloku.

Primer klase sa statičkim metodom i statičkim atributom dat je na slici 2.11.

```
class Staticka {
    static int brojStudenata = 0;
    static void inkrementirajBrojStudenata() {
        brojStudenata++;
    }
}
```

Slika 2.11. Korišćenje rezervisane reči this

U sledećem kodu inkrementiranje statičkog atributa izvedeno je pozivom statičke metode `inkrementirajBrojStudenata` pristupom preko objekata `objekat1` i `objekat2` te direktno na nivou klase `Staticka.`

```
Staticka objekat1 = new Staticka();
Staticka objekat2 = new Staticka();
objekat1.inkrementirajBrojStudenata();//brojStudenata=1
objekat2.inkrementirajBrojStudenata();//brojStudenata=2
Staticka.inkrementirajBrojStudenata();//brojStudenata=3
Staticka.brojStudenata++;             //brojStudenata=4
```

Statičke metode imaju pristup samo statičkim članovima klase.

2.4.3. Nizovi

Razlikuju se dva tipa nizova zavisno od tipa elemenata niza:

- elementi su primitivni tipovi;
- elementi su objekti.

Indeksiranje elementa unutar niza počinje od nule.Prvi element niza ima indeks nula.

Primer referenci na niz:

```
int[] vint; //referenca na niz int elemenata
Fen[] vfen; //referenca na niz Fen elemenata
```

Kreiranje (alokacija) niza na koji ukazuje referenca obavlja se korišćenjem oparatora `new`, npr. niz 10 celobrojnih vrednosti i niz 10 objekata klase `Fen`, respektivno:

```
vint = new int[10];
vfen = new Fen[10];
```

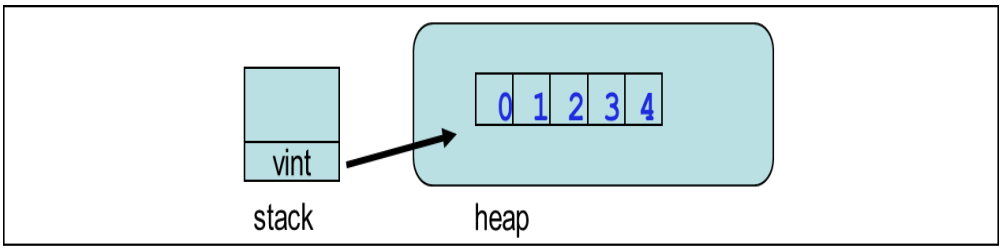
Dat je primer inicijalizacije vrednosti člana niza koji je primitivni tip (kao u C++):

```
vint[0]=1000;
```

Inicijalizacija niza se može izvesti i kao agregat (kao u C++):

```
int[] vint = { 0, 1, 2, 3, 4 };
```

Na slici 2.12. prikazano je stanje na stack-u i heap-u nakon ove inicijalizacije niza. Referenca na niz se čuva na steku, a pripadni elementi niza čuvaju se na heap-u.



Slika 2.12. Inicijalizacija niza čiji su elementi primitivnog tipa

Nizovi čiji su elementi objekti date klase zahtevaju još jedan korak u alokaciji memorije. Posmatraju se sledeće dve linije koda:

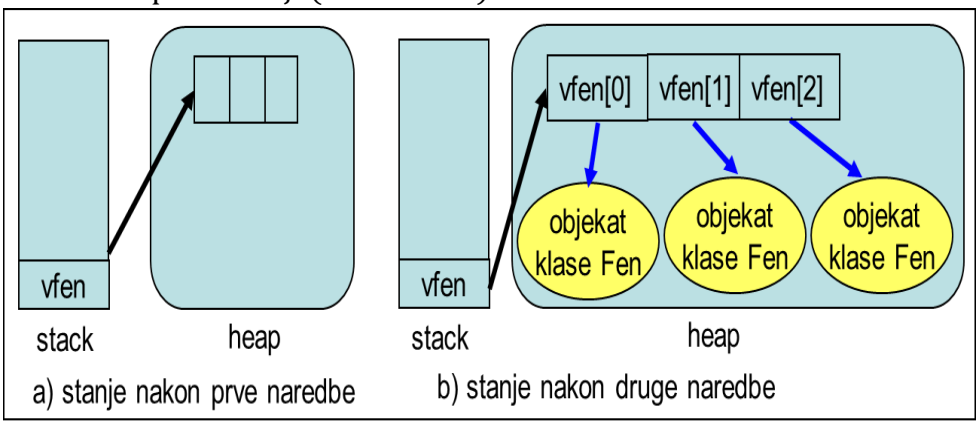
```
Fen[] vfen = new Fen[3];
```

```
for(int ix=0; ix<vfen.length; ix++) vfen[ix]=new Fen();
```

Nakon prve linije koda kreira se referenca `vfen` na stack-u koja ukazuje na blok memorije u heap-u u koji mogu da se smeste 3 reference na objekte klase `Fen` (slika 2.13.a).

Druga linija koda u `for` petlji menja indeks `ix` od nula do isključno broja `vfen.length`. Broj `vfen.length` pokazuje za koliko elemenata je niz `vfen` kreiran.

U petlji se inicijaliziraju reference na objekte klase Fen koje su smeštene u bloku heap memorije tako da sada svaka od njih ukazuje na svoj objekat klase Fen koji se nalazi u heap memoriji (slika 2.13.b).



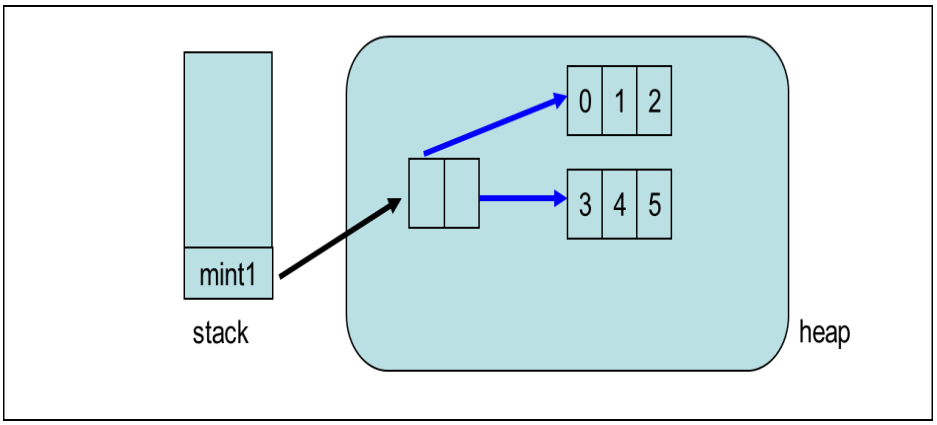
Slika 2.13. Kreiranje niza objekata klase Fen

Višedimenzionalni nizovi su nizovi čiji su elementi nizovi. Kreiraju se tri dvodimenzionalna niza (matrice 2x3) koji sadrže niz tipa `int`.

```
int[][] mint1 = { {0,1,2}, {3,4,5} };
int[][] mint2 = new int [2][3];
int[][] mint3 = new int [2][];
for (int i=0; i<mint3.length; i++) mint3[i]= new int[3];
```

Prva linija kreira, alocira i inicijalizuje dvodimenzionalni niz sa elementima redom od 0 do 5, na koga upućuje referenca `mint1` na heap-u.

Referenca `mint1` ukazuje na blok u heap memoriji gde su smeštene dve reference, svaka za po jedan red matrice. Ove dve reference ukazuju na pripadni blok u heap memoriji gde su smešteni elementi pripadnog reda matrice. U prvom bloku su respektivno elementi 0, 1, 2, a u drugom bloku su elementi 3, 4, 5. Pripadno stanje memorije je dato na slici 2.14.



Slika 2.14. Stanje memorije nakon kreiranja dvodimenzionalnog int niza

Druga linija kreira i alocira prostor za niz `mint2` bez inicijalizacije elemenata kao u prvoj liniji koda.

Treća i četvrta linja koda rade isto što i druga linija koda. Kreira se i alocira dvodimenzionalni int niz `mint3`, ali u dva koraka.

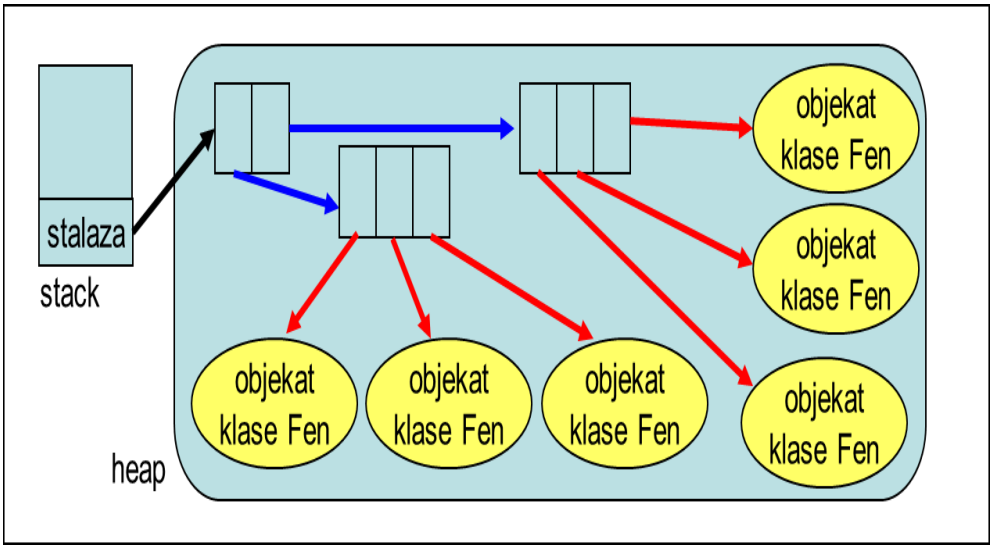
U trećoj liniji kreirana je referenca `mint3` na stack-u i blok memorije na heapu za smeštanje referenci koje će ukazivati na dva reda matrice.

Četvrta linija koda inicijalizuje reference na redove matrice tako da one sada ukazuju na blokove memorije za smeštaj int elemenata.

Kreira se dvodimenzionalni niz čiji su elementi nizovi referenci na objekte klase Fen, npr. fenovi raspoređeni na stalaži koja ima dve police gde na svaku policu staju tri fena.

```
Fen[][] stalaza = new Fen [2][];
for (int i = 0; i < stalaza.length; i++) {
  stalaza [i] = new Fen [3];
  for (int j = 0; j < stalaza[i].length; i++)
  stalaza [i][j] = new Fen ();
}
```

Slično primeru sa dvodimenzionalnim int nizom kreira se i dvodimenzionalni niz objekata gde je još potrebno izvršiti i kreiranje svakog objekta koji se nalazi u dvodimenzionalnom nizu (slika 2.15).



Slika 2.15. Dvodimenzionalni niz čiji su elementi objekti

Ekvivalentan rezultat se može dobiti korišćenjem agregata:

```
Fen[][] stalaza = {
                    { new Fen(), new Fen(), new Fen() },
                    { new Fen(), new Fen(), new Fen() }
                  };
```

gde se vrši kreiranje, alokacija i inicijalizacija dvodimenzionalnog niza objekata klase Fen.

Sledeći primer ilustruje korišćenje niza `String`-ova kao formalnog argumenta metode `main` koji pokazuje kako se ispisuju parametri poziva `main` metode koji se navode u komand promptu (slika 2.16).

```
class PrimerEhoParametara {
      public static void main(String[] argumenti) {
            for (int i=0; i<argumenti.length; i++)
                  System.out.print(argumenti[i]+" ");
      }
}
```

Slika 2.16. Ispis argumenta pri pozivu Java programa iz komandne linije

Neka je poziv prevedene date klase iz command prompta:

```
                java PrimerEhoParametara par1 par2 par3
rezultat je:
                par1 par2 par3
```

Razmak je postignut konkatenacijom sa jednim znakom za prazninu. Korišćena je metoda `print` koja ispisuje svoj stvarni argument na standardnom izlazu bez

prelaska u novi red.

Iteracija niza može se realizovati:

- konvencionalno:

```
public void ispisi(String[] niz){
      for (int i=0; i<niz.length;i++)
            System.out.println(niz[i]);
}
```

- iteracijom for-each, koja prolazi celim nizom (kolekcijom) i koristi tekući element niza koji se u primeru smešta u promenljivu `str`:

```
public void ispisi(String[] niz){
      for (String str: niz)
            System.out.println(str);
}
```

Obe iteracije ispisuju sve elemente niza poređane u koloni gde je svaki element niza prikazan u novom redu.

2.4.4. Nasleđivanje

Nasleđivanje klasa u Javi obavlja se klauzulom `extends` pri čemu je dozvoljeno eksplicitno nasleđivanje samo jedne klase.

Implicitno svaka klasa nasleđuje klasu `Object` čak i ako se ne navede da data klasa nasleđuje neku drugu klasu.

Klasa `Object` je u korenu hijerarhije nasleđivanja i nije apstraktna.

U ovom delu prikazuju se neke metode klase `Object`, a u nastavku knjige biće obrađene i druge veoma značajne metode ove klase.

Metoda `equals` poredi reference objekata:

```
public boolean equals(Object obj);
```

Neka postoje objekti `obj1` i `obj2` neke klase, tada poziv metode equals:

```
obj1.equals(obj2))
```

vraća rezultat određen implementacijom metode `equals` klase kojoj pripada objekat `obj1`.

Metoda `hashCode` izračunava hash vrednost datog objekta (kodni broj objekta):

```
public int hashCode();
```

Metoda `toString`() (pomenuta kod konkatenacije stringova) se koristi kod implicitne konverzije kada metoda čiji je formalni argument String dobija referencu na objekat neke klase. Tada se vrši poziv metoda `toString` te klase. Ako programer nije sam napisao nadjačao metodu `toString` biće pozvana podrazumevana metoda `toString` klase `Object`. U ovom slučaju biće ispisan

tekst koji se odnosi na naziv klase, zatim znak @ i onda heksadecimalno napisan heš kod objekta čija je metoda toString pozvana:

npr. `Automobil@1b6d3586,`

ili `getClass().getName()+'@'+Integer.toHexString(hashCode())`

U suprotnom slučaju, pozvaće se nadjačana metoda `toString` za dati objekt koju je programer napisao u okviru klase ovog objekta:

```
public String toString();
```

```
class Planeta{
   String naziv;
   Planeta(String nazivplanete){naziv=strNazivPlanete;}
   public String toString(){return naziv;}
 }
```

Slika 2.17. Primer korišćenja metode toString

Klasa `Planeta` ima podatak član naziv koji je refrenca na `String` (slika 2.17). Konstruktor klase `Planeta` ima formalni argument referencu na objekat klase `String`. Pozivom ovog konstruktora datoj planeti će biti dodeljen naziv. Ako se kao stvarni argument nekoj metodi koja očekuje referencu na `String` prosledi referenca na objekat klase `Planeta` biće implicitno pozvana nadjačana metoda `toString`:

```
Planeta pla = new Planeta("SATURN");
System.out.println("Naziv planete je "+pla+".");
```

Posmatra se primer u kom klasa `Zena` nasleđuje klasu `Osoba` (slika 2.18):

```
class Osoba {
      public String strIme;
      public String strPrezime;
      public void ispisiPodatke() { ... }
 }
 class Zena extends Osoba {
   public String strDevojackoPrezime;
   public void ispisiPodatke() { ... }
 }
```

Slika 2.18. Primer korišćenja metode toString

Klasa *`Zena`* eksplicitno nasleđuje, odnosno, proširuje klasu *`Osoba`* i implicitno klasu *`Object`*. Iza klauzule `extends` navodi se naziv osnovne klase (superklase). Sada objekat klase `Zena` pored podataka za devojačko prezime poseduje i podatke ime i prezime jer je `Zena` istovremeno i *`Osoba`*. Za razliku od C++ sve metode u Javi za koje nije navedeno da su final su virtuelne, odnosno, mogu da se nadjačaju, odnosno, redefinišu (*method overriding*) u klasi naslednici.

Na slici 2.19. dat je primer redefinisanja metode.

```
class A {
   void prva() { System.out.println("prva() iz A");  }
   void druga(){ System.out.println("druga() iz A"); }
}
class B extends A {
  void druga() { System.out.println("druga() iz B"); }
  public static void main(String []args){
    A a = new A();
    B b = new B();
    a.prva();
    b.prva();
    a.druga();
    b.druga();
  }
}
```

Slika 2.19. Primer redefinisanja metode

Prevođenjem i pokretanjem programa B izlaz će biti:

```
prva() iz A
prva() iz A
druga() iz A
druga() iz B
```

jer klasa B nasleđuje metodu `prva()` iz klase A, a metoda `druga()` je redefinisana u klasi B.

2.4.4.1. Apstraktne klase i metode

Metoda čija implementacija nije navedena predstavlja apstraktnu metodu što se označava ključnom reči apstrakt ispred njenog naziva. Na isti način, ako se ključna reč apstrakt nađe ispred imena klase, onda ova klasa ima bar jednu apstraktnu metodu.

Apstraktne klase ne mogu imati svoje objekte jer konstruktor ne bi mogao da kreira objekat čija je metoda neimplementirana. Na slici 2.20. dat je primer apstraktne klase.

```
public abstract class Apstraktna {
      private int i;
      public void metoda1() { ... }
      public abstract void metoda2();
}
```

Slika 2.20. Primer apstraktne klase

Apstraktne klase se koriste kada se želi napraviti generalizacija, a onda se u klasama naslednicama implementiraju apstraktne metode iz osnovne klase čime se postiže specijalizacija.

2.4.4.2. Interfejsi

Pošto Java može eksplicitno naslediti samo jednu klasu, problem implementacije višestrukog nasleđivanja rešen je uvođenjem interfejsa. Ispred naziva interfejsa navodi se ključna reč `interface` bez ključne reči `abstract`. U interfejsu se ne navodi ključna reč `abstract`. Interfejsi u Javi 9 mogu imati: konstante, apstraktne metode, podrazumevane metode, statičke metode, privatne metode i privatne statičke metode. Interfejs može naslediti drugi interfejs.

Klasa implementira interfejs implementacijom svih njegovih (apstraktnih) metoda. Navođenjem ključne reči implements iza naziva klase i ispred naziva interfejsa data klasa je u obavezi da implementira sve apstraktne metode navedenog interfejsa. Primer interfejsa i klase koja ga implementira dat je na slici 2.21.

```
interface KokpitAutomobila {
      void koristiMenjac();
      void koristiVolan();
}
class Ford implements KokpitAutomobila {
      void koristiMenjac () { //konkretno ponasanje ... }
      void koristiVolan () { //konkretno ponasanje ... }
}
```

Slika 2.21. Klasa Ford implementira interfejs KokpitAutomobila

Dozvoljeno je da klasa implementira više interfejsa (iza reči implements sledi popis interfejsa koji se implementiraju gde je delimiter zarez).

U Javi 9 podržano je korišćenje privatnih metoda interfejsa. Neka je dat primer na slici 2.22. Postavljanjem default metoda u interfejsu koji je definisan omogućuje se da se ista metoda odmah može koristiti u klasi koja implementira interfejs (ne mora se implementirati).

```
public interface Vozilo {
    String DEKOR="**********************";
    String brze();
    String sporije();
    default String ukljuciSportMod() {
        return "Ukljucen sport mod.";
    }
    default String iskljuciSportMod() {
```

```
        return "Iskljucen sport mod.";
    }
    default void autori(String koautor){
        imeautora();
        System.out.println("Koautor " + koautor);
    }
    static void KWuKS(double kw) {
        System.out.println(DEKOR);
        prefiks();
        System.out.println(ks +" --> "+(ks*1.34));
        System.out.println(DEKOR);
    }
    private void imeautora() {
        System.out.println("Autor: Pera");
    }
    private static void prefiks() {
        System.out.println("KW --> u KS");
    }
}
public class Automobil implements Vozilo {
    @Override
    public String brze() {
        return "Jurim brze.";
    }
    @Override
    public String sporije() {
        return "Jurim sporije.";
    }
    public static void main(String[] args) {
    Vozilo bmw = new Automobil("BMW");
    System.out.println(bmw.brze());
    System.out.println(bmw.sporije());
    System.out.println(bmw.ukljuciSportMod());
    System.out.println(bmw.iskljuciSportMod ());
    System.out.println(bmw.autori("Zika"));
    System.out.println(Vozilo.KWuKS(1));
   }
}
```

Slika 2.22. Implementacija interfejsa sa različitim tipovima metoda

U klasi Automobil koja implementra interfejs Vozilo definisane su metode brze i sporije, dok se ostale metode (default i static) direktno koriste. Pozivi privatnih metoda interfejsa se koriste unutar interfejsa.

Ako bi klasa implementirala dva interfejsa I1 i I2 pri čemu oba imaju iste default metode npr. m(), onda bi se razrešenje imena u toj klasi izvelo pozivom I1.super.m(), odnosno, I2.super.m(). Statička metoda interfejsa pripada interfejsu (ne objektu) tako da se poziv obavlja sa npr. Vozilo.KWuKS(1).

2.4.4.3. Unutrašnje klase

Unutrašnja klasa (*inner class*) predstavlja klasu koja je definisana unutar druge, spoljašnje klase (slika 2.23).

```
class Spo {
      void metodaSpoljasnje() { ... }
      class Unu {
            int metodaUnutrasnje() { ... }
      }
}
```

Slika 2.23. Unutrašnja klasa

Da bi se kreirao objekat unutrašnje klase mora se prvo kreirati objekat spoljašnje klase:

```
Spo s = new Spo();
Spo.Unu u = s.new Unu();
```

Nije dozvoljeno kreiranje objekta unutrašnje klase bez objekta spoljašnje klase.

Unutrašnje klase biće korišćene u poglavlju koje obrađuje grafički korisnički interfejs.

2.4.5. Polimorfizam

Polimorfizam omogućuje da prosleđivanjem stvarnog argumenta, koji je referenca na objekat izvedene klase, metodi čiji je formalni argument referenca na objekat osnovne klase bude pozvana istoimena metoda date izvedene klase.

Ovim mehanizmom objekti mogu da ispolje različito ponašanje, zavisno od njihove klase, bez obzira što se oni koriste kao instance nekog zajedničkog roditelja.

U sledećem primeru klase: `Ford` i `Formula1` izvedene su iz osnovne apstraktne klase `Menjac`. Ideja je da se polimorfizam iskoristi pozivom metode `saltaj` za slučaj da se probna vožnja obavlja fordovim vozilom ili formulom 1.

Klasa `ProbnaVoznja` sadrži metodu `probajMenjac` koja pozivom metode `saltaj` aktivira polimorfizam. Kodovi navedenih klasa dati su na slici 2.24.

```
abstract class Menjac {
      abstract void saltaj();
}
```

```
class Ford extends Menjac {
  void saltaj() {
       System.out.println("saltanje rucnim menjacem");
  }
}
class Formula1 extends Menjac{
  void saltaj() {
    System.out.println("saltanje tiptronik menjacem");
  }
}
      class ProbnaVoznja {
            void probajMenjac(Menjac menjac) {
                  menjac.saltaj();
            }
            public static void main(String []args){
                  Ford mondeo = new Ford();
                  Formula1 formula1 = new Formula1();
                  ProbnaVoznja pv = new ProbnaVoznja();
                  pv.probajMenjac(mondeo);
                  pv.probajMenjac(formula1);
                  pv.probajMenjac( new Ford() );
            }
      }
```

Slika 2.24. Polimorfizam

Apstraktna klasa `Menjac` sadrži apstraktnu metodu `saltaj()`. Klase `Ford` i `Formula1` nasleđuju klasu `Menjac` i implementiraju metodu `saltaj()`.

U datom primeru klasa `ProbnaVoznja` ima metodu `probajMenjac` koja kao parametar ima referencu na objekat klase `Menjac` pri čemu je ova klasa apstraktna i ne može imati objekte.

Ono što je dozvoljeno je da se kao stvarni argument metodi `probajMenjac` prosledi referenca na objekat klase koja je izvedena iz klase `Menjac`.

Pozivom metode `probajMenjac` sa stvarnim argumentom `mondeo` koji je referenca na objekat klase `Ford` koja je naslednica klase `Menjac` biće pozvana upravo redefinisana metoda `saltaj` klase Ford gde se šaltanje izvodi ručnim menjačem.

Analogno, prosleđivanjem stvarnog argumenta `formula1` pozvaće se upravo redefinisana metoda `saltaj` klase Formula1 gde se šaltanje izvodi prekidačima za menjanje brzina postavljenim na upravljaču.

2.4.6. Mehanizam obrade izuzetaka

U toku izvršavanja programa mogu se pojaviti greške koje je potrebno obraditi. Mehanizam obrade izuzetaka trebalo bi da obezbedi:

- da se informacija o izuzetku prenosi nezavisno od mehanizma prenosa argumenata i vraćanja vrednosti funkcije;
- da informacija o izuzetku bude bilo kog tipa;
- struktura programa na mestu obrade izuzetka treba da bude precizno determinisana na deo koji predstavlja osnovni tok programa i deo koji obuhvata kod za obradu izuzetka.

Ako se izuzetak desi tok programa automatski prelazi na deo za obradu izuzetka.

Deo programskog koda za koji smatramo da može da izazove izuzetak smešta se u `try/catch` blok. `Try` bloku može se pridružiti i više `catch` blokova (slika 2.25).

```
try {
      // kod u kom se moze desiti izuzetak
}
catch (ArithmeticException exception) {
      System.out.println("Izuzetak je:"+ exception);
}
catch (Exception ex) {
      System.out.println("Izuzetak je:"+ exception);
}
```

Slika 2.25. Blok za obradu izuzetaka

U `try` bloku se smešta kod koji može izazvati izuzetak. Ako se u ovom bloku desi izuzetak kontrola toka se premešta na početak `catch` blokova.

U datom primeru navedena su dva catch bloka:

- prvi catch ispituje da li se desio aritmetički izuzetak i ako jeste tok programa je takav da se izvrši kod u prvom catch bloku, a onda se preskaču svi ostali catch blokovi (u datom primeru još jedan catch) i nastavlja se izvršavanje koda iza catch bloka.
- ako se desio izuzetak koji nije aritmetički prvi catch blok se preskače i izvršiće se drugi catch blok koji prihvata obradu bilo kog izuzetka. Nakon izvršavanja koda u ovom catch bloku, program nastavlja rad odmah iza svih preostalih catch-ova ako ih ima.

U okviru catch bloka informacije o nastalom izuzetku dostupne su preko reference na objekat klase `Exception` ili klase naslednice `Exception`-a.

Daju se neke klase naslednice klase Exception;

- `ArithmeticException` odnosi se na sve izuzetke prilikom izvršavanja

aritmetikih operacija (deljenje nulom, prekoračenje,...);

- `ArrayIndexOutOfBoundsException` odnosi se na izuzetke kod pristupa elementu čiji je indeks van granice niza
- `NumberFormatException` odnosi se na neuspešan pokušaj konverzije npr. objekat klase `String` ne reprezentuje int vrednost.

U opštem slučaju kada u `try` bloku nastane izuzetak pripadni niz `catch` blokova se sekvencijalno ispituje i ulazi se u `catch` blok čiji formalni argument odgovara izuzetku koji se dogodio. Ako u redosledu `catch` blokova poslednji `catch` blok bude sa formalnim argumentom koji je referenca na objekat klase `Exception`, onda bi redosled obrade izuzetaka bio takav da bi se na kraju obrađivali svi izuzeci koji nisu navedeni kao argumenti prethodnih `catch` blokova. Ako se u `try` bloku ne desi izuzetak preskaču se svi pridruženi catch blokovi i ide se na kod u bloku finally (slika 2.26). Blok finally se ne mora eksplicitno navoditi.

```
try {
      // kod u kom se moze desiti izuzetak
}
catch (ArithmeticException ex) {
      System.out.println("Aritmetički izuzetak");
}
catch (ArrayIndexOutOfBoundsException ex) {
      System.out.println("Indeksiranje van granica niza");
}
catch (Exception ex) {
      System.out.println("Svi ostali izuzeci");
}
finally {
      // kod koji se izvršava desio se izuzetak ili ne
}
```

Slika 2.26. Blok try-catch-finally

Programer može kreirati svoju klasu izuzetaka koja mora naslediti klasu Exception (slika 2.27).

```
      public class MojIzuzetak extends Exception {
            public MojIzuzetak () {
                  super();
            }
            public MojIzuzetak (String msg) {
                  super(msg);
            }
      }
```

Slika 2.27. Korisnička klasa koja proširuje klasu Exception

U ovom primeru metoda `super()` koja je navedena u konstruktorima klase `MojIzuzetak` poziva konstruktore superklase `Exception`, bez argumenata i sa referencom na objekat klase `String`, respektivno. Korišćenjem klauzule `throw` izaziva se izuzetak (slika 2.28):

```
try {
      if (uslovZaIzuzetak())
      throw new MojIzuzetak("Korisnicki izuzetak!");
}
catch (MojIzuzetak mi) {
       System.out.println("Exception: " + mi);
}
```

Slika 2.28. Korišćenje klauzule throw

Moguće je eksplicitno navesti iza imena metode da ona može izazvati dati izuzetak (slika 2.29). Pozivanje ove metode je smešteno u `try` blok ili ako je metoda pozvana iz druge metode, onda ta druga metoda mora biti označena da može izazvati izuzetak ili je u `try` bloku.

```
public void prvaMetoda() throws MojIzuzetak{
      if (uslovZaIzuzetak())
      throw new MojIzuzetak("Korisnicki izuzetak!");
}
public void drugaMetoda() {
      try {  prvaMetoda ();  }
      catch (MojIzuzetak ex) {
            System.out.println("Exception: " + ex);
      }
}
public void trecaMetoda() throws MojIzuzetak {
     prvaMetoda ();
}
```

Slika 2.29. Korišćenje klauzule throws

Ovim mehanizmom je omogućena propagacija obrade izuzetka sve do metode `main`.

2.4.7. Kloniranje

Operator dodele samo će napraviti duplikat reference na objekat. Ono što se želi je da se napravi duplikat objekta.
Metoda Object.clone() omogućuje dupliranje objekta. Pravo pristupa ovoj metodi je protected. Referenca na klonirani objekat se razlikuje od reference na original, dok su klase originala i kopije iste:

```
a.clone() != a
a.clone().getClass() == a.getClass()
```

Da bi se definisala metoda `clone()` klasa mora da implementira interfejs `Cloneable` (spada u takozvani marker ili tag interfejs koji nema članove, informacija za JVM). Ako prethodno nije zadovoljeno, generiše se izuzetak `CloneNotSupportedException`, a ako je prethodna provera pozitivna, onda će se kreirati i vratiti kao rezultat plitka kopija objekta (*shallow copy*, *bit-wise* kopija objekta).

Shallow copy kreira kopije atributa što je korektno ako su u pitanju primitivni tipovi. Problem je pri kopiranju atributa koji predstavljaju reference na objekte (pri kopiranju original i kopija će se odnositi na isti objekat na heap-u).
Rešenje je da se uradi duboko kloniranje (*deep cloning*) tako da će i 'drugi' nivo kloniranja biti uključen.

Neka je data klasa `Course` koja se odnosi na kurs koji ima 3 predmeta (slika 2.30) i služi kao kontejner. Druga klasa koja se koristi u primeru je klasa `Student` koja implementira interfejs `Cloneable` (slika 2.31). Klasa `Student` opisuje stanje studenta sa: id studenta, imenom studenta i nazivom kursa koga student pohađa. Nadjačana metoda `clone()` radi plitko kloniranje objekta. Klasa `ShallowCopyInJava` kreira kurs `science` sa tri predmeta te dva studenta: `student1` i `student2` (slika 2.32). `Student1` je referenca koja gleda na objekat `Student` kreiran pozivom pripadnog konstruktora. `Student2` je referenca koja je definisana kao referenca na plitku kopiju objekta na koga gleda referenca `student1`. Ako se sada izvrši promena 3. predmeta za studenta `student2` desiće se i promena 3. predmeta studenta `student1`.

```
class Course {
    String subject1;
    String subject2;
    String subject3;
    public Course(String sub1, String sub2, String sub3)    {
        this.subject1 = sub1;
        this.subject2 = sub2;
        this.subject3 = sub3;
    }
}
```

Slika 2.30. Klasa Course

```
class Student implements Cloneable{
    int id;
    String name;
    Course course;
    public Student(int id, String name, Course course)  {
        this.id = id;
        this.name = name;
        this.course = course;
```

```
    }
    //podrazumevana metoda clone(), plitka kopija objekta
    protected Object clone() throws CloneNotSupportedException{
        return super.clone();
    }
}
```

Slika 2.31. Klasa Student

```
public class ShallowCopyInJava {
    public static void main(String[] args)    {
        Course science = new Course("Physics", "Chemistry",
                                                    "Biology");
        Student student1 = new Student(111, "John", science);
        Student student2 = null;
        try
        {
           //klon studenta student1 dodeljuje se studentu student2
           student2 = (Student) student1.clone();
        }
        catch (CloneNotSupportedException e)
        {
            e.printStackTrace();
        }
        //stampanje predmeta subject3 studenta student1
        System.out.println(student1.course.subject3);
        //Output : Biology
        //promena predmeta subject3 studenta student2
        student2.course.subject3 = "Maths";
        //promena se refelektuje na original : student1
        System.out.println(student1.course.subject3);
        //Output : Maths
    }
}
```

Slika 2.32. Klasa ShallowCopyInJava

Ono što se želi je da se prilikom kloniranja dobiju dva nezavisna objekta. Sledi primer duboke kopije gde je potrebno na oba nivoa postaviti pravilno kloniranje. Sada je potrebno da klasa kursa nazvana `Course2` takođe implementira kloniranje (slika 2.33). Klasa `Student2` sada u metodi `clone()` klonira tekućeg studenta, ali i posebno njegov kurs (slika 2.34). Ako se sada iz klase `DeepCopyInJava2` izvrši kloniranje, prilikom promene 3. predmeta kloniranog studenta (objekta) neće doći do uticaja na originalnog studenta jer se radi o dva nezavisna objekta (slika 2.35).

```
class Course2 implements Cloneable{
    String subject1;
```

```
    String subject2;
    String subject3;
    public Course2(String sub1, String sub2, String sub3)    {
        this.subject1 = sub1;
        this.subject2 = sub2;
        this.subject3 = sub3;
    }
    protected Object clone() throws CloneNotSupportedException{
        return super.clone();
    }
}
```

Slika 2.33. Klasa Course2

```
class Student2 implements Cloneable{
    int id;
    String name;
    Course2 course2;
    public Student2(int id, String name, Course2 course2)    {
        this.id = id;
        this.name = name;
        this.course2 = course2;
    }
    //nadjacana metoda clone() za kreiranje duboke kopije objekta
    protected Object clone() throws CloneNotSupportedException
    {
        Student2 student2 = (Student2) super.clone();
        student2.course2 = (Course2) course2.clone();
        return student2;
    }
}
```

Slika 2.34. Klasa Student2

```
public class DeepCopyInJava2{
    public static void main(String[] args)    {
        Course2 science = new Course2("Physics","Chemistry",
                                                "Biology");
        Student2 student11 = new Student2(111, "John", science);
        Student2 student22 = null;
        try{
            student22 = (Student2) student11.clone();
        }
        catch (CloneNotSupportedException e)  {
            e.printStackTrace();
        }
        System.out.println(student11.course2.subject3);
        //Output:Biology
```

```
        student22.course2.subject3 = "Maths";
        System.out.println(student11.course2.subject3);
        //Output:Biology
    }
}
```

Slika 2.35. Klasa DeepCopyInJava2

2.5. Inicijalizacija statičkih članova

Ranije je objašnjeno da statičko polje ima samo jednu instancu koja pripada klasi. Polje je na nivou klase, a ne objekta čime nije potrebno kreirati konkretan objekat da bi se pristupilo statičkom polju. Inicijalizacija statičkog polja se prva obavlja, dakle pre korišćenja bilo kog statičkog polja i pre izvršavanja bilo kog metoda klase (2.36).

```
    class Inicijalizacija{
      public static int sx = 10;
      private int y;
      public void postaviy() {
        y=2*sx;  //ovde se zna vrednost sx
      }
    }
```

Slika 2.36. Inicijalizacija statičkog polja

Statička polja mogu se inicijalizovati u statičkim inicijalizacionim blokovima (slika 2.37). Redosled statičke inicijalizacije je sleva-udesno i odozgo-naniže.

```
class PrimerStatInit{
  public static int[] nizFibonaci=new int[50];
  static {
      popuniFibonacijevNiz();
  }
  private static void popuniFibonacijevNiz(){
    nizFibonaci[0]= nizFibonaci[1]=1;
    for (int i=2; i< nizFibonaci.length; i++)
      nizFibonaci[i]= nizFibonaci[i-1] +
                      nizFibonaci[i-2];
  } //probajte rekurzivno u jednoj liniji koda
  public static void main(String arg[]){
    System.out.println("Peti element Fibonacijevog niza je " +
                        PrimerStatInit.nizFibonaci[4]);
  }
}
```

Slika 2.37. Statički inicijalizacioni blok

U primeru je u metodi `main` dat pristup statičkom polju preko naziva klase i operatora pristupa članu. Statički metod može direktno pristupati samo statičkim poljima i statičkim metodima klase. Pošto je statički metod na nivou klase nema reference `this`. Fibonačijev niz ima prva dva elementa jednaka 1, a svaki sledeći element je jednak zbiru svoja dva prethodnika (1, 1, 2, 3, 5, 8, ...).

2.5.1. Problem ciklične statičke inicijalizacije

Sledi ilustracija rešenja problema ciklične statičke inicijalizacije (slika 2.38). Neka postoji po jedan statički inicijalizator u klasama `A` i `B`. Ako svaki statički inicijalizator poziva metod iz one druge klase može se desiti da npr. `B` još nije inicijalizirana dok se `A` inicijalizuje.

```
class A{
  static {
     System.out.println("stat. blok A: pocelo izvrsavanje");
     B.metodB();
     System.out.println("stat. blok A: zavrsava izvrsavanje");
  }
  static void metodA(){ System.out.println("metodA"); }
}
class B{
  static {
     System.out.println("stat. blok B: pocelo izvrsavanje");
     A.metodA();
     System.out.println("stat. blok B: zavrsava izvrsavanje");
  }
  static void metodB(){ System.out.println("metodB");}
}
class PrimerStatCiklInic{
  public static void main(String[] args){
     A a = new A();
  }
}

Izlaz:
       stat. blok A: pocelo izvrsavanje
       stat. blok B: pocelo izvrsavanje
       metodA
       stat. blok B: zavrsava izvrsavanje
       metodB
       stat. blok A: zavrsava izvrsavanje
```

Slika 2.38. Statički inicijalizacioni blok

U datom primeru navedeni problem ciklične statičke inicijalizacije je rešen

hronološki na sledeći način:

- inicijalizator klase `A` izvršava se do poziva metoda `B.metodB()`
- inicijalizator klase `B` izvršava se do poziva metoda `A.metodA()`
- izvršava se pozvani metod klase `A`
- nastavi se izvršavanje inicijalizatora klase `B`
- izvršava se pozvani metod klase `B`
- nastavi se izvršavanje inicijalizatora klase `A`

2.5.2. Statički atribut kao referenca na objekat pripadne klase

Slede dva povezana primera u kojima se prikazuje korišćenje statičkog atributa, nasleđivanja i poziva metoda superklase iz pripadnog nadjačanog metoda (slika 2.39)

```
import java.util.*;

class Tacka{
  private double x;
  private double y;
  public static Tacka ishodiste=new Tacka();
  public Tacka() {inicijalizuj();}
  public Tacka(double x, double y){
    this.x=x;
    this.y=y;
  }

  public void inicijalizuj(){
    x=0;y=0;
  }

  public double rastojanje(Tacka t){
    double dx, dy;
    dx=x-t.x; dy=y-t.y;
    return Math.sqrt(dx*dx+dy*dy);
  } //moze i Math.hypot(dx,dy)

  public static void main(String args[]){
    //testiranje klase
    Tacka A = new Tacka(100.0,100.0);
    System.out.println("Udaljenost tacke A od ishodista je "
                       + Tacka.ishodiste.rastojanje(A) );
    //ili
    System.out.println("Udaljenost tacke A od ishodista je "
                       + A.rastojanje(Tacka.ishodiste) );
    Tacka B = new Tacka();
    System.out.println("Udaljenost tacke B od ishodista je "
```

```
                        + Tacka.ishodiste.rastojanje(B) );
  }
}
```

Slika 2.39. Statički atribut kao referenca na objekat pripadne klase

U datom primeru klasa `Tacka` ima statički atribut `ishodiste` koji je refrenca na objekat iste klase i predstavlja jedinstveni element koji pripada celoj klasi. Ovim će biti kreirana jedna kopija ishodišta za sve objekte klase.

Prikazani su pozivi metoda `rastojanje` gde je stvarni argument referenca na objekat klase `Tacka` te atribut klase `ishodiste`, respektivno.

Na slici 2.40. dato je proširenje klase `Tacka`.

```
class Tacka3D extends Tacka{
  private double z;     //z koordinata
  public static Tacka3D ishodiste3D=new Tacka3D();
  public Tacka3D() {inicijalizuj();}
  public Tacka3D(double x, double y, double z){
    this.x=x;
    this.y=y;
    this.z=z;
  }
  public double rastojanje3D(Tacka3D t){
    double dx, dy, dz;
    dx=x-t.x; dy=y-t.y; dz=z-t.z;
    return Math.sqrt(dx*dx+dy*dy+dz*dz);
  } //ili Math.hypot(dx,dy);
  public void inicijalizuj(){
    super.inicijalizuj();
    z=0;
  }
}
```

Slika 2.40. Trodimenzionalna tačka

U primeru je redefinisan metod `inicijalizuj` koji se oslanja na deo posla koji će obaviti poziv metoda `inicijalizuj` superklase za x i y koordinatu te dodatni posao koji inicijalizuje z koordinatu.

2.6. Korišćenje paketa

U Javi postoje dva načina za korišćenje klase iz nekog paketa. Moguće je korišćenje celog naziva navođenjem navigacije ka klasi u okviru datog paketa i naziva klase ili uvođenjem paketa u fajl klase u kojoj će se koristiti naziv date klase iz tog paketa.

```
      class PunNaziv{
```

```
        public static void main(String[] args){
              java.util.Date datum=new java.util.Date();
              System.out.println(danas);
        }
}
```

Slika 2.41. Pun naziv klase koja se koristi

Na slici 2.41. koristi se klasa `Date` koja je nalazi u okviru paketa `java.util`. Ovaj paket sadrži standardnu biblioteku klasa. Navodi se pun naziv paketa u kom se klasa nalazi i sam naziv klase.

```
import java.util.Date;

class ImportKlase{
      public static void main(String[] args){
         Date danas = new Date();
         System.out.println(danas);
      }
}
```

Slika 2.42. Uvoz paketa kome pripada klasa koja se koristi

Na slici 2.42. je korišćenje klase `Date` koja je nalazi u okvitu paketa java.util. implementirano uvozom ove klase klauzulom `import java.util.Date`. Nakon ovog je dovoljno samo navesti naziv klase `Date` bez navođenja paketa u kom se klasa nalazi.

Ako bi se koristilo:

```
import java.util.*;
```

onda ovo znači uvoz svih klasa koje se nalaze u paketu `java.util`, ali ne i klasa koje se nalaze u potpaketima ovog paketa. Dozvoljeno je isključivo korišćenje znaka * za uključenje svih klasa datog paketa ili konkretnog naziva klase za uključenje te klase.

2.7. Moduli

U Javi 9 omogućen je modularni pristup pri razvoju aplikacija. Sve počinje od *Jigsaw* projekta koji postaje *Java Platform Module System (JPMS).* Ideja je bila da se softver dizajnira tako da je podeljen u module koji zajedno čine celinu. Modul predstavlja kolekciju paketa, klasa, interfejsa, kodova, podataka i resursa (modul je kontejner paketa).

Cilj je da modul bude funkcionalna celina koja može da deluje nezavisno od ostatka sistema čime se omogućuje:

- bolje performase aplikacije
- smanjenje veličine aplikacije što je podesno za manje uređaje
- smanjenje složenosti sistema

- lakše je testiranje i debagovanje
- lakše je održavanje koda
- efikasniji kod
- povećana je enkapsulacija itd.

Naziv modula mora biti jedinstven na nivou projekta. Pravila koja važe za imenovanje paketa, važe i kod imenovanja modula (ne koristiti donju crtu). Naziv modula mora biti jedinstven npr. konvencijski `com.pera.mojmodul`.

U korenskom direktorijumu svakog modula u folderu *src* mora postojati deklaraciona datoteka (`module-info.java`) u kojoj se navode specifikacije zavisnosti od drugih modula (navodi se reč `requires`) kao i specifikacije koji se modul može koristiti u drugim modulima (navodi se reč `exports`). Pri deklarisanju nije dozvoljeno napraviti konflikt ciklične zavisnosti (dva modula zahtevaju jedan drugog).

Osnovni modul koji je implicitan je modul `java.base`. Ovo znači da ovaj modul nije potrebno eksplicitno navoditi da se koristi u deklaraciji ostalih modula jer će biti automatski dodat pri kompilaciji. Modul `java.base` ne zahteva druge module (`requires`) već ima izvoz (`exports`) paketa kao što su: `java.lang`, `java.util`, `java.math`, `java.io`, `java.net` itd.

Neka se kreira projekat naziva npr. `PrimerKoriscenjaModula` u integrisanom razvojnom okruženju *JetBrains IntelliJ IDEA* (JDK mora biti minimalno 9, za prazan projekt u *Project Settings>Project* postaviti: *SDK* 9, *Project language level*: 9). U okviru projekta (*Project settings* > *Module*, znak "+") kreiraju se dva modula: `modul.pozdrav` i `modul.test`.

U modulu `modul.pozdrav` (u folderu `src`) kreira se paket `primer.pozdrav` *(*folder `primer/pozdrav`*)* u kome se kreira klasa `PozdravSvete`. Ova klasa sadrži metodu `pozdrav()` koja vraća string "`Hello world !`" (slika 2.43).

```
package primer.pozdrav;
public class PozdravSvete {
    public String pozdrav(){
        return "Zdravo svete";
    }
}
```

Slika 2.43. Klasa PozdravSvete

U modulu `modul.test` (u folderu `src`) kreira se paket `testiranje.pozdrav` *(*folder `testiranje/pozdrav`*)* u kome se kreira klasa `TestPozdrav`. Ova klasa sadrži metodu `main()` koja definiše referencu *ps* na objekat klase `PozdravSvete` i ispisuje povratnu vrednost poziva metode `pozdrav()`. Videti sliku 2.44.

```
package testiranje.pozdrav;
import primer.pozdrav.PozdravSvete;
```

```
public class TestPozdrav {
    public static void main(String[] args) {
        PozdravSvete ps=new PozdravSvete();
        System.out.println(ps.pozdrav());
    }
}
```

Slika 2.44. Klasa TestPozdrav

Modul `modul.test` zahteva korišćenje paketa `primer.pozdrav` modula `modul.pozdrav`. Ova zavisnost mora biti navedena u deklaracionim datotekama *(*`module-info.java`*)* oba modula. Dodavanje deklaracionih datoteka može se izvršiti klikom desnim tasterom računarskog miša na `src` folder unutar datog modula gde se u padajućem meniju bira opcija *New>module-info.java.*

Sadržaj deklaracione datoteke `module-info.java` modula `modul.pozdrav` dat je na slici 2.45. Navedena je dozvola za korišćenje klasa iz paketa `primer.pozdrav` u svim modulima koji zahtevaju ovaj modul.

```
module modul.pozdrav {
    exports primer.pozdrav;
}
```

Slika 2.45. Deklaraciona datoteka module-info.java modula modul.pozdrav

Sadržaj deklaracione datoteke `module-info.java` modula `modul.test` je dat na slici 2.46. U ovoj datoteci navedena je zavisnost modula `modul.test` od modula `modul.pozdrav`.

```
module modul.test {
    requires modul.pozdrav;
}
```

Slika 2.46. Deklaraciona datoteka module-info.java modula modul.test

Sada je potrebno u *JetBrains IntelliJ IDEA* projektu navesti i zavisnost modula preko opcija podešavanja da bi moduli bili povezali na nivou projekta. Klikne se desnim tasterom računarskog miša na naziv projekta, a onda odabere opcija *Open Module Settings***.** Otvara se prozor u kome se bira modul kome dodeljujemo zavisnost, a onda se u kartici *Dependencies* klikom na znak "+" odabere opcija *Module Dependency* i odabere potreban modul (*modul.pozdrav*).

2.8. Neke klase iz paketa java.util

U ovoj sekciji biće prikazane neke od često korišćenih klasa paketa java.util.

2.8.1. Klasa Vector

Klasa `Vector` je kontejnerska klasa kolekcije objekata kojima se pristupa

indeksno. Obezbeđeno je dodavanje i uklanjanje elemenata na proizvoljoj indeksnoj poziciji. Klasa koristi polimorfizam tako da smešta reference na objekte tipa `Object`, čime je postignuta univerzalnost u tipu elemenata koje klasa `Vector` može sadržavati. Objekat klase `Vector` može istovremeno sadržavati objekte različitih klasa (slika 2.47).

```
import java.util.Vector;

class PrimerVector {

  public static void main(String args[]) {
   Vector vector = new Vector();
    vector.addElement("Nulti element je String");
    vector.addElement(new Integer(5));
    vector.addElement("Drugi element ");
    for (int i=0; i<vector.size(); i++)
       System.out.println(vector.elementAt(i) + " ");
  }
}
```

Slika 2.47. Primer korišćenja klase java.util.Vector

U prethodnom primeru metoda `addElement` dodaje element u vektor, dok metoda `elementAt(i)` uzima element iz vektora čiji je indeks i. Kastovanje, odnosno, eksplicitno konvertovanje (casting), se radi da bi se ono što se uzme iz vektora tretiralo na pravi način budući da se u vektor mogu smeštati bilo koji objekti.

2.8.2. Klasa Hashtable

Klasa `Hashtable` vrši mapiranje ključeva u vrednosti. Ključevi i vrednosti su objekti. U primeru koji sledi prikazuje se izvlačenja kuglica sa brojevima iz bubnja sa ponavljanjem (izvučena kuglica se ponovo vraća u bubanj).

Klasa `Brojac` služi za pamćenje koliko je puta izvučena data kuglica sa brojem (slika 2.48). Konstruktor ove klase inicijalno dodeljuje podatku članu `i` vrednost 1 koja označava je je kuglica kojoj je pridružen objekat ove klase upravo izvučena.

```
// napomena: klasu Brojac snimiti u fajl Brojac.java

class Brojac{
  private int i;
  Brojac(){
    i=1;
  }

  void Inkrementiraj(){
    i++;
  }
```

```
  public String toString() {
    return Integer.toString(i) + "\n";
  }
}
```

Slika 2.48. Klasa Brojac

```
// napomena: klasu PrimerHashtable snimiti u fajl
//           PrimerHashtable.java
import java.util.*;
class PrimerHashtable{
  public static void main(String args[]) {
    Hashtable ht = new Hashtable();
    for (int i = 0; i < 10000; i++) {
      Integer r = new Integer((int)(1+(Math.random()*39)));
      if(ht.containsKey(r))
         ((Brojac)ht.get(r)).Inkrementiraj();
      else        ht.put(r, new Brojac());
    }
    System.out.println(ht);
  }
}
```

Slika 2.49. Klasa PrimerHashtable

Klasa `PrimerHashtable` u svojoj main metodi kreira referencu na objekat klase `Hashtable` (slika 2.49). Ovde nisu korišćeni generički tipovi (radi se kasnije).

U petlji od 10000 iteracija kreira se referenca na objekat klase `Integer` čijem konstruktoru se šalje slučajan broj od 1 do 39.

Proverava se metodom `containsKey` da li `hash`-tabela ima ključ koji odgovara tom broju:

- ako ne postoji takav ulaz on se kreira metodom `put` koja ima formalne argumente ključ i vrednost, a to su ovde `Integer r` i novi neimenovani `Brojac`. Ovde će konstruktor klase `Brojac` postaviti vrednost brojanja na 1.
- ako postoji ulaz za dati ključ, znači da je dati broj već izvučen, sledi uzimanje tog brojača metodom `get` i inkrementiranjem njegove vrednosti brojanja. Uzimanje elementa je rešeno korišćenjem kastovanja na objekat koji se uzima.

2.8.3. Klasa StringTokenizer

Klasa `StringTokenizer` parsira string prema datom graničniku (slika 2.50)

```
import java.util.*;

class PrimerStringTokenizer {

  public static void main(String args[]) {
    String text = "Token prvi#drugi token#treci token";
    StringTokenizer st = new StringTokenizer(text, "#");
    while (st.hasMoreTokens()) {
         System.out.println(st.nextToken());
    }
  }
}
```

Slika 2.50. Klasa PrimerHashtable

U prethodnom primeru biće ispisani delovi stringa, tokeni, koji su razdvojeni graničnikom koji je zadat kao znak #. Metodom `hasMoreTokens` proverava se da li postoji bar jedan token za parsiranje koga vraća metoda `nextToken`.

2.8.4. Klasa ArrayList

Klasa `ArrayList` realizuje listu nizom promenljive dužine. Niz sadrži reference na objekte. Dužina ovog niza se menja dinamički stavljanjem i uklanjanjem elemenata niza. Ovu listu je moguće kreirati: kao praznu (`ArrayList()`), sa početnom veličinom niza (`ArrayList( int velicinaniza)`) ili inicijalizovanu kolekcijom (`ArrayList(Collection kolekcija)`).

Na slici 2.51. dat je ilustrativni primer korišćenja liste u obliku niza u kome se dodaju i uklanjaju elementi te ispisuje veličina i sadržaj liste.

```
import java.util.*;

class PrimerArrayList{

  public static void main(String args[]) {
    ArrayList arraylist = new ArrayList();
    arraylist.add("Ovo je ");
    arraylist.add("B");
    arraylist.add("M");
    arraylist.add("W");
    arraylist.add(" 2019");
    arraylist.add("Beograd");
    arraylist.add(0, "->");
    arraylist.remove("Beograd");
    arraylist.remove(1);
    System.out.println("Velicina: " + arraylist.size());
    System.out.println("Sadrzaj : " + arraylist);
```

```
    Object arrObj[] = arraylist.toArray();
    for(int i=0; i<arrObj.length; i++)
      System.out.print( arrObj[i] );
  }
}

Program ce ispisati:
   Velicina: 5
   Sadrzaj : [->, B, M, W,  2019]
   ->BMW 2019
```

Slika 2.51. Primer korišćenja klase ArrayList

U ovom primeru korišćena su metode: `add` za dodavanje elementa, `remove` za uklanjanje elementa, `size` za uzimanje veličine liste u obliku niza i `toArray` za pretvaranje liste u obliku niza u običan niz `Object` elemenata.

2.8.5. Klasa LinkedList

Klasa `LinkedList` realizuje povezanu listu. Na slici 2.52. dat je primer koji ilustruje korišćenje ove klase.

```
import java.util.*;

class PrimerLinkedList{

   public static void main(String args[]) {
      LinkedList linkedlist = new LinkedList();
      linkedlist.add("B");
      linkedlist.add("f");
      linkedlist.add("M");
      linkedlist.add("W");
      linkedlist.add(0, "je ");
      linkedlist.addFirst("Ovo ");
      linkedlist.addLast("Beograd");
      linkedlist.addLast("2019");
      linkedlist.remove("f");
      linkedlist.remove(1);
      linkedlist.removeFirst();
      linkedlist.removeLast();
      Object vrednost = linkedlist.get(1);
      linkedlist.set(1,(String) vrednost + "-M-M");
      System.out.println("Sadzaj :"+linkedlist);
   }
}
```

Slika 2.52. Primer korišćenja klase LinkedList

```
Program ispisuje:
   Sadzaj :[B, M-M-M, W, Beograd]
```

U datom primeru metodama: `add` - dodaje se element u povezanu listu, `addFirst` – dodaje se element na početak povezane liste, `addLast` – dodaje se element na kraj povezane liste, `get` – uzima se indeksirani element povezane liste, `getFirst` – vraća prvi element povezane liste, `getLast` – vraća poslednji element povezane liste, remove – uklanja indeksirani ili asocirani element povezane liste, `removeFirst` – uklanja prvi element povezane liste, `removeLast` – uklanja poslednji element povezane liste, `set` – postavlja vrednost indeksiranog elementa povezane liste.

2.8.6. Klasa TreeSet

Klasa `TreeSet` realizuje stablo gde se Objekti smeštaju po rastućem redosledu. Moguće je kreirati prazno stablo (`TreeSet()`), stablo koje će se popuniti kolekcijom (`TreeSet(Collection kolekcija)`), prazno stablo koje se uređuje prema komparatoru (`TreeSet(Comparator komparator)`) ili stablo koje će biti uređeno (`TreeSet(SortedSet uredjenskup)`). Na slici 2.53. dat je ilustrativni primer korišćenja klase `TreeSet`.

```
import java.util.*;

class PrimerTreeSet{
  public static void main(String args[]) {
    TreeSet treeset = new TreeSet();
    treeset.add(new Integer(6));
    treeset.add(new Integer(3));
    treeset.add(new Integer(5));
    treeset.add(new Integer(2));
    treeset.add(new Integer(1));
    treeset.add(new Integer(4));
    System.out.println(treeset);
  }
}

Ispisace se:
[1, 2, 3, 4, 5, 6]
```

Slika 2.53. Primer korišćenja klase TreeSet

2.8.7. Iteratori

Često je potrebno proći redom kroz celu kolekciju za šta se koristi interfejs `Iterator`. Svaka klasa kolekcija metodom `iterator()` vraća iterator koji pokazuje na početak kolekcije. Na slici 2.54. dat je primer korišćenja iteratora.

```
import java.util.*;

class PrimerIterator{

  void ispisi(ArrayList arraylist, String naslov){
```

```
      Iterator iterator = arraylist.iterator();
      System.out.println(naslov);
      while(iterator.hasNext()) {
      Object element = iterator.next();
            System.out.print(element + ",");
      }
      System.out.println();
  }
  public static void main(String args[]) {
      PrimerIterator primer = new PrimerIterator();
      ArrayList arraylist = new ArrayList();
      arraylist.add("B");
      arraylist.add("M");
      arraylist.add("W");
      primer.ispisi(arraylist,"prvi ispis:");
      ListIterator listiterator = arraylist.listIterator();
      while(listiterator.hasNext()) {
      Object element = listiterator.next();
            listiterator.set(element+
                        ((String)(element)).toLowerCase());
      }
      primer.ispisi(arraylist,"drugi ispis:");
      System.out.println("ispis unazad:");
      while(listiterator.hasPrevious()) {
            Object element = listiterator.previous();
            System.out.print(element + ",");
      }
  }
}

Ispisace se:
   prvi ispis:
   B,M,W,
   drugi ispis:
   Bb,Mm,Ww,
   ispis unazad:
   Ww,Mm,Bb,
```

Slika 2.54. Primer korišćenja iteratora

U datom primeru popunjava se lista znacima "B","M","W" respektivno, a onda se poziva metoda ispisi objekta klase PrimerIterator. Ova metoda ima dva formalna argumenta, prvi je referenca na objekat klase ArrayList dok je drugi referenca na objekat klase String. Metoda ispisi kreira Iterator za prosleđeni ArrayList, pozivom njegove metode iterator. Ovaj iterator

pokazuje na početak date kolekcije.

U while petlji proverava se uslov da li postoji element u listi metodom `hasNext` te ako postoji uzima se referenca na isti metodom `next` te ispisuje na konzoli. Sada iterator ukazuje na sledeći element liste. Klasa `ListIterator` ima mogućnost da se kroz listu prolazi u oba smera korišćenjem parova metoda `hasNext`, `next` i `hasPrevious`, `previous`.

2.8.8. Klasa HashSet

Klasa `HashSet` koristi se kada je potrebno elemente staviti u skup. Ovim se svaki element može samo jednom dodati u skup. Ovo pravilo važi i za `null` element.

```
import java.util.*;

public class HashSetPrimer {

    public void hashSetExample() {
        Set vehicles = new HashSet();
        //elementi koji ce biti stavljeni u skup
        String item_1 = "Ford";
        String item_2 = "BMW";
        String item_3 = "Audi";
        boolean result;
        //dodavanje elemenata u skup
        result = vehicles.add(item_1);
        System.out.println(item_1 + ": " + result);
        result = vehicles.add(item_2);
        System.out.println(item_2 + ": " + result);
        result = vehicles.add(item_3);
        System.out.println(item_3 + ": " + result);
        //pokusaj dodavanja istog elementa
        result = vehicles.add(item_1);
        System.out.println(item_1 + ": " + result);
        //dodavanje null elementa
        result = vehicles.add(null);
        System.out.println("null: " + result);
        //ponovno dodavanje null elementa
        result = vehicles.add(null);
        System.out.println("null: " + result);
    }

    public static void main(String[] args) {
        new HashSetPrimer().hashSetExample();
    }
}
```

```
Izlaz:
      Ford: true
      BMW: true
      Audi: true
      Ford: false
      null: true
      null: false
```

Slika 2.55. Primer korišćenja klase HashSet

Na slici 2.55. dat je primer korišćenja klase `HashSet`. Metodom `add` dodaju se elementi, ali će uspešno biti dodati samo elementi koji su jedinstveni u skupu, što vredi i za element `null`.

2.8.9. Klasa HashMap

Klasa `HashMap` ne garantuje redosled elemenata u smislu da redosled unesenih i redosled pročitanih elemenata mape korišćenjem iteratora ne mora biti isti. Vreme izvršavanja osnovnih operacija (`put` i `get`) ostaje isto i pri velikim količinama unesenih elemenata.

```
import java.util.*;

class HashMapPrimer {

    public static void main(String args[]) {
        // kreiranje HashMap-e
        HashMap hm = new HashMap();
        // stavljanje elemenata u mapu
        hm.put("Elvis Presley", new Double(7650376.00));
        hm.put("Tom Jones",
                new Double(6650381.50));
        hm.put("Tina Turner", new Double(9870023.75));
        // kreiranje skupa ulaza
        Set set = hm.entrySet();
        // uzimanje iterarora
        Iterator i = set.iterator();
        // prikaz elemenata
        while (i.hasNext()) {
            Map.Entry me = (Map.Entry) i.next();
            System.out.print(me.getKey() + ": ");
            System.out.println(me.getValue());
        }
        System.out.println();
        // dodavanje na racun 10000
        double balance = ((Double) hm.get
```

```
                        ("Tina Turner")).doubleValue();
        hm.put("Tina Turner",
                new Double(balance + 10000));
        System.out.println("Tina Turner, novo stanje: "
                           + hm.get("Tina Turner"));
    }
}
Izlaz:
   Tina Turner: 9870023.75
   Tom Jones: 6650381.5
   Elvis Presley: 7650376.0
   Tina Turner, novo stanje: 9880023.75
```

Slika 2.56. Primer korišćenja klase HashMap

Na slici 2.56. dat je primer korišćenja klase `HashMap`. Metodom `put` se dodaje asocijativna veza objekata ključ-vrednost. Metoda `entrySet` vraća kolekciju (skup) ulaza. Uzimanjem iteratora skupa ulaza u mapu metodom `iterator` omogućeno je kretanje kroz mapu. Referenca `me` na objekat klase `Map.Entry` dobija se pozivom metode `next` iteratora uz kastovanje na `Map.Entry`. Sada se za dati ulaz mogu dohvatiti ključevi, metodom `getKey` te pripadne vrednosti metodom `getValue`. Standardni način da dobijemo vrednost za dati ključ je da se koristi metoda mape get kojoj se kao pramatar prosledi vrednost koja odgovara ključu. Pošto nisu korišćeni generici potrebno je kastovanje vraćene vrednosti (zarada Tine Turner). Pozivanjem metode put za vrednost ključa koji već postoji uradiće se ažuriranje (update) vrednosti na koju se ključ odnosi (bonus za Tinu).

Na slici 2.57. dat je primer koji pokazuje korišćenje liste korisničkih objekata.

```
//datoteka RandR.java
class RandR{
   private String ime;
   private String prezime;
   private int pozicija;
  RandR (String ime, String prezime, int pozicija) {
     this.ime      = ime;
     this.prezime  = prezime;
     this.pozicija = pozicija;
   }
   public String toString() {
      return ime + " " + prezime + ", " + pozicija;
   }
}
```

```
//datoteka PrimerRockAndRoll.java
import java.util.*;

class PrimerRockAndRoll{

   public static void main(String args[]) {
      LinkedList linkedlist = new LinkedList();
      linkedlist.add(new RandR ("Elvis","Presley"    , 1));
      linkedlist.add(new RandR ("Tom"  ,"Jones"      , 2));
      linkedlist.add(new RandR ("Bruce","Springsteen", 3));
      Iterator iterator = linkedlist.iterator();

      while(iterator.hasNext()) {
         Object element = iterator.next();
         System.out.println(element);
      }
      System.out.println();
   }
}

Izlaz:
   Elvis Presley, 1
   Tom Jones, 2
   Bruce Springsteen, 3
```

Slika 2.57. Primer liste korisničkih objekata

U jednostavnom primeru data je klasa `PrimerRockAndRoll` koja u metodi `main` kreira povezanu listu objekata tipa `RandR`. Klasa `RandR` predstavlja jednostavan kontejner čiji su atributi ime, prezime i pozicija (na rang listi) i ima nadjačanu metodu `toString` za korisnički ispis podataka o objektu. Korišćenjem iterarora navedene liste ispisuju se članovi iste liste korišćenjem nadjačane metode `toString`.

2.8.10. instanceof

Rezervisana reč `instanceof` čitajte kao: da li referenca ukazuje na konkretan objekat date klase (ne na `null`). Na slici 2.58. dat je primer u kome se pokazuje korišćenje `instanceof`.

```
public class MainClass {
   public static void main(String[] a) {
      String s = "Hello";
      int i = 0;
      String g = null;
      if (s instanceof java.lang.String) {
         System.out.println("s is a String");
      }
      //if (i instanceof Integer) {
      //   System.out.println("i is an Integer");
```

```
      //} //nece dozvoliti ovu proveru
      if (g instanceof java.lang.String) {
         System.out.println("g is a String");
      }
}
Izlaz je:
   s is a String
```

Slika 2.58. Primer instanceof

Program daje izlaz da referenca `s` pokazuje na objekat klase `String`. Referenca `g` ne gleda na objekat klase `String` već na `null`. Da biste probali proveru sa identifikatorom `i` , umesto `int i` probajte `Integer i` .

2.9. Rad sa ulazom i izlazom

Rad sa ulazom u izlazom biće prikazan kroz konkretne primere koji se odnose na upisivanje u izlaz te čitanje iz ulaza. Biće prikazani načini rada sa datotekama i standardnim ulazom (tastaturom).

2.9.1. Klase File, FileReader, FileWriter

Sledi primer program koji kopira sadržaj datoteke `ulazna.txt` u datoteku `izlazna.txt` korišćenjem klasa `File`, `FileReader`, `FileWriter`. Pri radu sa datotekama može nastati izuzetak `IOException`, tako da je to u primeru datom na slici 2.59. rešeno navođenjem da main može izazvati isti (`throws IOException`).

```
import java.io.*;

public class Copy {

  public static void main(String[] args) throws IOException{
    File inputFile = new File("ulazna.txt");
    File outputFile = new File("izlazna.txt");
    FileReader in = new FileReader(inputFile);
    FileWriter out = new FileWriter(outputFile);
    int c;
     while ((c = in.read()) != -1){
         out.write(c);
     }
     in.close();
     out.close();
    }
}
```

Slika 2.59. Kopiranje iz fajla u fajl

Klase za rad sa ulazom i izlazom u datom primeru sadržane su u paketu

java.io. Konstruktoru klase File prosleđen je stvarni argument koji predstavlja naziv datoteke.

Klase FileReader i FileWriter služe za čitanje i upis podataka u datoteku čija je relevantna referenca na objekat tipa File prosleđena kao stvarni parametar konstruktora ovih klasa.

Metoda read() klase FileReader učitaće karakter sa tekuće pozicije iz ulazne datoteke, dok će metoda write(c) upisati karakter c na tekuću poziciju u izlaznu datoteku. Obe metode pomeraju tekuću poziciju pripadne datoteke. Kada se stigne do kraja ulazne datoteke povratna vrednost metode read() biće -1.

Zatvaranje otvorenih datoteka vrši se metodom close() pripadne klase FileReader ili FileWriter. U datom primeru metoda main izaziva IOException ako neka od operacija nad datotekom ne uspe.

2.9.2. Klasa RandomAccessFile

Za korišćenje slučajnog pristupa datoteci može se koristiti klasa RandomAccessFile. U sledećem primeru kreira se datoteka u koju se upisuju i čitaju različiti tipovi podataka (slika 2.60).

```
import java.io.File;
import java.io.RandomAccessFile;
import java.io.IOException;

public class DemoRandomAccessFile {

  private static void probaRAFa() {
    try {
      File file = new File("C:/primer/RAFdemo.out");
      RandomAccessFile raf =new RandomAccessFile(file,"rw");
      raf.writeByte(65);
      raf.writeBytes("Strbac");
      raf.write(0x0A);
      raf.seek(0);
      // cita se karakter
      byte ch = raf.readByte();
      System.out.println("Prvi karakter u fajlu je : " +
                         (char)ch);
      // sada se cita preostali deo linije
      // od trenutne pozicije kursora
      // sve do znaka za kraj linije
      System.out.println("Linija teksta je : " +
                          raf.readLine());
      // pomeranje na kraj fajla
```

```
      long mojapozicija = file.length();
      raf.seek(mojapozicija);
      // dodavanje na kraj fajla
      raf.write(0x0A);
      raf.writeBytes("Zavrsni tekst.");
      // pozicioniranje na pocetak dodatog dela
      raf.seek(mojapozicija);
      System.out.println("Dodato je:"+ raf.readByte() +
                         raf.readLine());
      raf.close();
    }
    catch (IOException e) {
            System.out.println("IOException:");
            e.printStackTrace();
    }
  }
  public static void main(String[] args) {
      probaRAFa();
  }
}

Izlaz:
   Prvi karakter u fajlu je : A
   Linija teksta je : Strbac
   Dodato je:10Zavrsni tekst.
```

Slika 2.60. Korišćenje klase RandomAccessFile

U prethodnom primeru kreirana je referenca file na klasu File koja se odnosi na datoteku `C:/primer/RAFdemo.out`. Referenca raf na objekat klase `RandomAccessFile` omogućuje baratanje navedenom datotekom (preko reference `file`) pri čemu je u konstruktoru dat i drugi parametar koji se odnosi na način baratanja `"rw"` što znači da će se navedena datoteka koristiti i za čitanje i za upis. U datoteku se upisuje bajt čija je decimalna vrednost 65 metodom `writeByte(65),` zatim bajtovi koji se odnose redom na znakove stringa metodom `writeBytes("Strbac"),` nakon ovoga upisuje se heksadecimalna vrednost `0x0A` metodom `write(0x0A)` i na kraju se kursor datoteke pomera na početak datoteke metodom `seek(0).` Sada se vrši čitanje prve dve upisane vrednosti metodama: `readByte`() za čitanje upisanog bajta (65 je kod za slovo A), i `readLine`() za čitanje upisanog stringa do znak za kraj reda (`0x0A`). Vrši se pomeranje kursora datoteke na kraj datoteke tako da se metodi seek prosledi vrednost koja odgovara dužini datoteke koja se dobija sa `file.length`(). Ova pozicija se pamti u long promenljivoj `mojapozicija`. Dodaje se bajt `0x0A` i string `"Zavrsni tekst"` na kraj datoteke. Sledi čitanje vrednosti sa zapamćene pozicije od koje su upisani navedeni bajt i string. Bajt `0x0A` je

decimalno jednak `10`. Na kraju se datoteka zatvara metodom `close()`. Kod koji može da baci izuzetak je smešten u `try-catch` blok u kome se očekuje izuzetak tipa `IOException`.

2.9.3. Klase DataOutputStream, FileOutputStream, DataInputStream, FileInputStream

Sledeći primer demonstrira korišćenje parova klasa `DataOutputStream`, `FileOutputStream` i `DataInputStream`, `FileInputStream` (slika 2.61). Pojedinačno učitavanje (`import`) potrebnih klasa obavilo je razvojno okruženje.

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class WriteBinary2 {
      public static void main(String[] argv)
                              throws IOException {
            int i=42;
            double d = Math.PI;
            String strFileName ="binbin.bin";
            DataOutputStream dos =
                 new DataOutputStream(
                     new FileOutputStream(strFileName));
            dos.writeInt(i);
            dos.writeDouble(d);
            dos.close();
            DataInputStream dis =
                new DataInputStream(
                    new FileInputStream(strFileName));
            System.out.println("procitan : "+ dis.readInt() +
                               "\nprocitan : " +
                               dis.readDouble());
            dis.close();
  }
}
Izlaz:
   procitan : 42
   procitan : 3.141592653589793
```

Slika 2.61. Upis i čitanje datoteke pomoću klasa DataInputStream, FileInputStream, DataOutputStream, FileOutputStream

U primeru se vrši binarni upis vrednosti promenljivih `i` i `d` (tipa `int` i `double`) u

datoteku "binbin.bin" korišćenjem dos reference na objekat tipa DataOutputStream čijem konstruktoru je prosleđena referenca na objekat tipa FileOutputStream. Konstruktoru klase FileOutputStream je prosleđena referenca strFileName koja se odnosi na string koji sadrži naziv datoteke (binbin.bin). Pozivom metoda writeInt i writeDouble upisuju se vrednosti i i d u datoteku "binbin.bin", respektivno. Zatvara se datoteka metodom close(), a onda se korišćenjem reference dis na objekat tipa DataInputStream čijem je konstruktoru prosleđena referenca na objekat tipa FileInputStream čitaju upisane vrednosti pozivom metoda readInt() i readDouble(), respektivno. Konstruktoru klase FileInputStream je prosleđena referenca strFileName.

2.9.4. Klasa Scanner

Za unos sa tastature koristi se klasa Scanner ili stream-ovi koji će biti korišćeni u poglavlju TCP/IP komunikacije. Na slici 2.62. dat je primer korišćenja klase Scanner.

```
import java.util.*;
public class Skener {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        scan.useDelimiter("\\n"); // dobro za String pomocu next()
        System.out.println("unesite integer ? ");
        int i = scan.nextInt();   // unos integera
        System.out.println("integer je " + i);
        System.out.println("unesite long ? ");
        long ll = scan.nextLong(); // unos longa
        System.out.println("long je " + ll);
        System.out.println("unesite string ? ");
        String stri = scan.next(); // unos Stringa
        System.out.println("integer je " + stri);
    } // napisite bolji kod
}
```

Slika 2.62. Klasa Scanner

Stvarni argument koji se prosleđen konstruktoru klase Scanner je System.in što govori da se očekuje standardni sistemski ulaz (tastatura). Metoda nextInt() preuzima unos sa tastature i tretira ga kao celobrojnu vrednost. Za double tip podataka bila bi metoda nextDouble() i slično (za long bila bi metoda nextLong()).

Za preuzimanje unosa sa tastature i tretiranja istog kao String može se koristiti i nextLine() (ali paziti na zaostali \n, tako da se iza unosa int, double, long ubaci jedan nextLine()).

```
Scanner sc = new Scanner(new File("myNumbers"));
```

```
      while (sc.hasNextLong()) {
          long aLong = sc.nextLong();
      }
```

Slika 2.63. Čitanje datoteke pomoću klase Scanner

Ako bi se `Scanner` koristio za učitavanje iz datoteke npr. "`myNumbers`" u koju su upisani `long` brojevi, onda bi sintaksa bila kao na slici 2.63.

2.9.5. Klase InputStreamReader i BufferedReader

Sledeći kod se odnosi na korišćenje klasa `InputStreamReader` i `BufferedReader` (slika 2.64)

```
import java.io.*;
public class IOHelp {
  private static InputStreamReader isr = new
                 InputStreamReader(System.in);
  private static BufferedReader br = new
                 BufferedReader(isr);
  public static String readLine(String p) {
      String retVal = "";
      System.out.print(p+"> ");
      try {
        retVal = br.readLine();
      }
      catch (Exception e){
        System.out.println("IOHelp: " + e.getMessage());
      }
      return retVal;
  }
  public static int readInt(String prompt) {
    try {
      return Integer.parseInt(readLine(prompt));
    }
    catch(Exception e) {
      System.out.println("Error reading int");
      return 0;
    }
  }
  public static double readDouble(String prompt) {
    try {
      return Double.parseDouble(readLine(prompt));
    }
    catch(Exception e) {
      System.out.println("Error reading double");
```

```
      return 0.0;
    }
  }
}
```

Slika 2.64. Korišćenje klasa InputStreamReader i BufferedReader

Par `InputStreamReader` i `BufferedReader` koriste se da se preuzmu podaci iz ulaznog streama, a to je u primeru `System.in`.

Konstruktor klase `BufferedReader` očekuje da mu se prosledi referenca na objekat klase `InputStreamReader`. Za kreiranje reference na `InputStreamReader` objekat potrebno je proslediti konstruktoru ulazni stream.

Metoda `readLine()` klase `BufferedReader` vraća uzetu liniju teksta sa ulaza. Povratna vrednost ove metode koristi se u metodama `readInt` i `readDouble` kao stvarni argument za metode `parseInt` i `parseDouble` metoda klase `Integer` i `Double` respektivno. Metoda `readLine()` može izazvati izuzetak pa je potrebno okruženje `try-catch` blokom.

Sledi trivijalan primer korišćenja klasa `InputStreamReader` i `BufferedReader` koji proverava parnost učitanog prirodnog broja.

```
import java.io.*;
class TestParan{
  public static void main( String[] args){
    InputStreamReader isr = new
                            InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(isr);
    int i;
    String str ="";
    try {
      do{
          System.out.println("unesite prirodan broj ");
          str = br.readLine();
          i = Integer.parseInt(str);
      }while(i<1);
      switch(i%2){
        case 0:System.out.println(i+" je paran broj");break;
        default:System.out.println(i+" je neparan broj");
      }
    }
    catch (Exception e){
      System.err.println("IOHelp: " + e.getMessage());
    }
  }
```

```
}
```

Slika 2.65. Provera parnosti učitanog prirodnog broja

U sledećem primeru potrebno je uneti broj između 1 i 5 što predstavlja ulaz za rekurzivnu metodu koja rešava problem kula Hanoja (slika 2.66).

Legenda kaže da je indijski bog Brahma prilikom stvaranja sveta postavio tri dijamantna štapa, a na prvom štapu 64 zlatna koluta različitog prečnika pri čemu je manji kolut na većem kolutu. Potrebno je prebacivati kolut po kolut, ali tako da se ne sme staviti veći kolut na manji kolut pri čemu se drugi (srednji) štap koristi kao pomoćni, a novu kulu kolutova treba dobiti na trećem štapu (po legendi tada nastupa kraj sveta).

Ideja je jednostavna:

- prebaciti za 1 kolut manju kulu sa prvog na drugi štap,
- preostali najveći kolut sa prvog štapa prebaciti na treći štap
- manju kulu sa drugog štapa prebaciti na treći štap koristeći prvi štap kao pomoćni.
- grana rekurzije staje kada u grani rekurzije nema kolutova (n=0).

```
import java.io.*;
public class Hanoj{
  private void prebaci(int n, int sa, int na, int pom){
        if(n>0){
          prebaci(n-1,sa,pom,na);
          System.out.println(sa+"-->"+na);
          prebaci(n-1,pom,na,sa);
        }
  }
  public static void main(String args[]){
   Hanoj han = new Hanoj();
   InputStreamReader isr=  new InputStreamReader(System.in);
   BufferedReader br = new BufferedReader(isr);
   String str="";
   int brojkolutova;
   do{
       System.out.println("unesite broj kolutova?(1-5)");
       try{
       str = br.readLine();
       brojkolutova = Integer.parseInt(str);
     }
     catch( Exception ex) {
       System.out.println( "Izuzetak "+ex);
```

```
        brojkolutova = -1;
      }
    }while((brojkolutova<1)||(brojkolutova>5));
    han.prebaci(brojkolutova,1,3,2);
   }
}

Prikaz na konzoli:
    unesite broj kolutova?(1-5)
    4
    1-->2
    1-->3
    2-->3
    1-->2
    3-->1
    3-->2
    1-->2
    1-->3
    2-->3
    2-->1
    3-->1
    2-->3
    1-->2
    1-->3
    2-->3
```

Slika 2.66. Kule Hanoja

2.9.6. Serijalizacija

Postupak serijalizacije je konvertovanje stanja objekta u niz bajtova koji se snima u npr. datoteku. Modifikator `transient` određuje da promenljiva kojoj je pridružen ovaj modifikator neće biti sačuvana u postupku serijalizacije. Sve promenljive objekta podrazumevano nisu `transient`, tako da ako se u procesu serijalizacije želi izbeći snimanje neke od ovih promenljivih mora se ispred date promenljive eksplicitno navesti modifikator `transient`. Da bi se obezbedila serijalizacija, mora se implementirati interfejs `Serializable` koji ne deklariše niti jednu metodu.

Primer serijalizacije u kojoj će se snimiti atributi objekta koji se odnose na tip vozila i kubikažu, dok se atribut koji se odnosi na marku vozila neće snimiti u datoteku (`automobili`) dat je na slici 2.67.

```
import java.io.*;

class SerijalizacijaPrimer implements Serializable{
    private transient String markaVozila;
    private String tipVozila;
```

```
    private  int kubikazaVozila;
    public SerijalizacijaPrimer (String markaVozila,
                         String tipVozila,
                         int kubikazaVozila){
        this.markaVozila = markaVozila;
        this.tipVozila = tipVozila;
        this.kubikazaVozila = kubikazaVozila;
    }
    public String toString(){
        StringBuffer sb = new StringBuffer(40);
        sb.append("\nMarka vozila: ");
        sb.append(this.markaVozila);
        sb.append("\nTip vozila: ");
        sb.append(this.tipVozila);
        sb.append("\nKubikaza vozila: ");
        sb.append(this.kubikazaVozila);
        sb.append(" kubika");
        return sb.toString();
    }
    public static void main(String args[])
                                          throws Exception {
        SerijalizacijaPrimer vozilo =
           new SerijalizacijaPrimer("Ford","Mustang",4000);
        ObjectOutputStream oos =
           new ObjectOutputStream(
               new FileOutputStream("automobili"));
        // upisivanje objekta
        oos.writeUnshared(vozilo);
        oos.close();
        // citanje objekta
        ObjectInputStream ois =
           new ObjectInputStream(
               new FileInputStream("automobili"));
        SerijalizacijaPrimer vozilo2 =
           (SerijalizacijaPrimer)ois.readObject();
        System.out.println(vozilo2);
    }
}

Izlaz:
   Marka vozila: null
   Tip vozila: Mustang
   Kubikaza vozila: 4000 kubika
```

Slika 2.67. Serijalizacija

U glavnoj metodi definiše se referenca `vozilo` na objekt klase `SerijalizacijaPrimer` (parametri: "`Ford`", "`Mustang`", `4000` za marku, tip i kubikažu vozila). Definiše se referenca `oos` na objekt klase `ObjectOutputStream` čijem konstruktoru se prodleđuje referenca na objekt klase `FileOutputStream`. Parametar konstruktora `FileOutputStream` je string "`automobili`" koji predstavlja datoteku u koju će se upisati objekt vozilo. Metodom `writeUnshared`(`vozilo`) upisuje se vozilo (tip i kubikaža) u datoteku "`automobili`", a onda se metodom `close()` zatvara izlazni tok. Sada se definiše referenca `ois` na objekt klase `ObjectInputStream` čijem konstruktoru se prodleđuje referenca na objekt klase `FileInputStream`. Parametar konstruktora `FileInputStream` je string "`automobili`" koji predstavlja datoteku iz koje će se pročitati objekt. Metodom `readObject()` čita se objekt iz datoteke "`automobili`", a onda se ispisuju njegovi podaci korišćenjem preklopljene metode `toString`. U metod to `String` dat je primer korišćenja klase `StringBuffer` (metoda `append` za dodavanje na kraj `StringBuffera` i `toString` za kreiranje stringa iz `StringBuffera`).

Rezime poglavlja Java

U poglavlju Java obrađeni su: tipovi podataka, kontrola toka programa, objektno orijentisani pristup, inicijalizacija statičkih članova, korišćenje paketa, koncepcija modula, klase iz biblioteke java.util, rad sa ulazom i izlazom.

Zadaci za proveru znanja iz poglavlja Java:

1. Napisati program koji ima klasu Tacka koja realizuje tačku u datoj dimenziji (2D, 3D, ... , nD). Ova klasa ima metodu koja vraća udaljenost između dve tačke istih dimenzija.
2. Napisati program koji simulira parking koji ima 2x30 parking mesta. Vozila vremenski nasumično dolaze i popunjavaju parking, odnosno, vremenski nasumično odlaze i oslobađaju parking.
3. Napisati program koji simulira loto izvlačenje.
4. Napisati program za auto salon koji vodi podatke o automobilima u datoteci (snimati serijalizovano).
5. Napisati program koji dati razlomak prikazuje kao egipatski. Egipatski prikaz razlomka je suma razlomaka kojima je brojnik jednak 1.
   Npr. $3/4 = 1/2 + 1/4$
   Pomoć:
   razlomak $3/4$ $ceil(1/(3/4)) = 2$ --> prvi sabirak je $1/2$ ostaje $3/4 - 1/2 = 1/4$
   razlomak $1/4$ $ceil(1/(1/4)) = 4$ --> drugi sabirak je $1/4$ ostaje $1/4 - 1/4 = 0$ kraj

3. KONKURENTNO PROGRAMIRANJE

Cilj poglavlja je da student ovlada: kreiranjem niti, raspoređivanjem niti, korišćenjem metoda sleep i yield, završavanjem niti, suspendovanjem i reaktiviranjem niti, sinhronizacijom niti, problemom deadlocka, sinhronizacije proizvođača i potrošača i korišćenjem modifikatora volatile.

Konkurentno programiranje odnosi se na pisanje programa u kome se u samom programu kreira bar jedna paralelna programska nit (thread) ili više paralelnih programskih niti (*Multithreaded programming*) koje nezavisno ili sinhronizovno izvršavaju svoj kod.

Pokretanjem bilo kog Java programa paralelno se odvijaju bar dve programske niti: glavna nit i garbage collector na koju programer nema uticaj pri čemu je programeru omogućeno da kreira nove programske niti. Način izvršavanje ovih niti određen je operativnim sistemom, hardverom i JVM-om, ali to programer ne vidi, tako da nisu potrebne izmene programa za izvršavanje na različitim računarskim platformama.

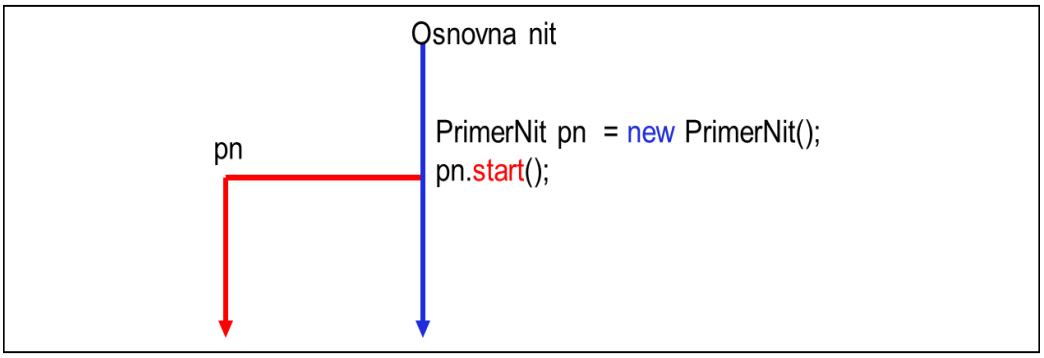
Da bi data klasa (ili njene naslednice) imala svojstvo niti potrebno je naslediti klasu `Thread`. U metodi `run` klase `Thread` smešta se kod koji se izvršava u paralelnoj niti (slika 3.1).

Paralelnost bi trebalo shvatiti uslovno, jer će se na jednoprocesorskoj mašini u jednom vremenskom intervalu izvršavati jedna nit, u drugom druga (ili možda ponovo ista) nit itd.

```
public class PrimerNit extends Thread {
  public void run() {
      System.out.println("run metoda u paralelnoj niti");
  }
  public static void main(String []args) {
      PrimerNit pn = new PrimerNit();
      pn.start();
  }
}
```

Slika 3.1. Kreiranje niti

Klasa `PrimerNit` nasleđuje klasu `Thread` čime nasleđuje i svojstvo niti. U main metodi kreira se referenca na objekt klase `PrimerNit`, a onda poziva njegova metoda `start` nasleđena iz klase `Thread`. Ovim pozivom metoda `start` obavlja potrebnu inicijalizaciju i poziva metodu `run` čime je nit pokrenuta. Sada se program sastoji od dve niti: osnovne niti programa koja počinje svoje izvršavanje od metode `main`, i nove niti klase `PrimerNit`. što je ilustrovano na slici 3.2.



Slika 3.2. Pokretanje nove niti

Naglašeno je da eksplicitno višestruko nasleđivanje nije dozvoljeno u Javi te da je problem rešen korišćenjem interfejsa. U tom smislu za kreiranje programske niti implementira se interfejs `Runnable` koji ima metodu `run` (slika 3.3).

```
public class PrimerNit2 implements Runnable{
  public void run() {
      System.out.println("run metoda u paralelnoj niti");
  }
  public static void main(String []args) {
      PrimerNit2 pn2a = new PrimerNit2();
      Thread    th    = new Thread(pn2a);
      th.start();
      Runnable pn2b = new PrimerNit();
      th  = new Thread(pn2b);
      th.start();
  }
}
```

Slika 3.3. Implementacija interfejsa Runnable

U datom primeru u metodi main kreirana je referenca na objekt klase `PrimerNit2`, a onda je kreirana referenca na objekt klase `Thread` gde je pozvan konstruktor sa parametrom kome je prenesena referenca na objekt klase `PrimerNit2`. Iza sledi poziv metode `start` kreiranog objekta klase `Thread`.

Drugi deo koda u metodi main pokazuje drugi način za kreiranje niti klase koja implementira interfejs `Runnable`. Obe niti se ponašaju kao i nit klase `PrimerNit` jer imaju isti programski kod u metodi `run`.

Postoje dva tipa niti u Javi:

- nit čiji završetak Java program ne čeka i to je demonska nit (*daemon),* kao što je garbage collector;

- nit na koju Java program čeka da se završi i to je nedemonska ili korisnička nit *(non-daemon).*

Proglašavanje niti za demonsku ili korisničku obavlja se pomoću metode `setDaemon`:

```
nit1.setDaemon(true);   // demonska
nit2.setDaemon(false);  // korisnicka
```

Tip niti se nasleđuje od roditeljske niti. Nakon startovanja niti tip niti se ne može menjati inače pri pokušaju promene desiće se izuzetak `IllegalStateException`.

Java program startuje sa jednom *non-daemon* (glavnom) niti koja počinje izvršavanje metode `main` i jednom *daemon* niti (*garbage collector*). Dalje je na programeru da kreira dodatne niti.

Prioritet izvršavanja niti može se postaviti metodom `setPriority(int)`. Formalni argument ove metode određuje nivo prioriteta u izvršavanju date niti. Vrednosti se kreću od 1 do 10, pri čemu se mogu koristiti i konstante klase `Thread: MIN_PRIORITY, NORM_PRIORITY i MAX_PRIORITY` koje imaju vrednosti 1, 5 i 10 respektivno. Redom su navedeni minimalni, normalni i maksimalni prioritet niti.

Nit inicijalno ima prioritet roditeljske niti. Prioritet niti može se menjati u bilo kom trenutku čak i ako se ista izvršava.

Metod `getPriority()` vraća prioritet niti.

Sledi primer u kom se kreiraju dve niti sa odgovarajućim prioritetima i svojstvima demonstva (slika 3.4).

```
class ImePrezime extends Thread {
  public String rec;
  public int pauza;
  public ImePrezime(String rec, int pauza) {
    this.rec = rec;
    this.pauza = pauza;
  }
  public void run() {
    while (true) {
      try {
        Thread thread = Thread.currentThread();
     System.out.println(rec+" Id = "+thread.getId());
        System.out.print(" "+rec+" Name = "+
            thread.getName());
```

```
        System.out.print(" "+rec+" Prioritet = "+
          thread.getPriority());
        System.out.print(" "+rec+
          (thread.isDaemon() ?
             " DEMONSKA " : " KORISNICKA "));
        sleep(pauza);
      }
      catch (Exception ex) {
        System.out.println("Desio se izuzetak"+ex);
      }
    }
  }
  public static void main(String[] args){
            ImePrezime ime    = new ImePrezime("Pera" ,1000);
            ImePrezime prezime= new ImePrezime("Peric",3000);
            ime.setPriority(Thread.MIN_PRIORITY);
            prezime.setPriority(Thread.MAX_PRIORITY);
            ime.setName("IME");
            ime.setDaemon(true);
            ime.start();
            prezime.setName("PREZIME");
            prezime.setDaemon(true);
            prezime.start();
            try{
                  sleep(5000);
            }
            catch (Exception ex)
         {
              System.out.println("Izuzetak u main: " +ex );
            }
      }
}
```

Izlaz:

```
Pera Id = 9
 Pera Name = IME Pera Prioritet = 1 Pera DEMONSKA Peric Id = 10
 Peric Name = PREZIME Peric Prioritet = 10 Peric DEMONSKA Pera Id
= 9
```

```
 Pera Name = IME Pera Prioritet = 1 Pera DEMONSKA Pera Id = 9
 Pera Name = IME Pera Prioritet = 1 Pera DEMONSKA Peric Id = 10
 Peric Name = PREZIME Peric Prioritet = 10 Peric DEMONSKA Pera Id
= 9
 Pera Name = IME Pera Prioritet = 1 Pera DEMONSKA Pera Id = 9
 Pera Name = IME Pera Prioritet = 1 Pera DEMONSKA
```

Slika 3.4. Dve niti sa odgovarajućim svojstvima demonstva

U prethodnom primeru kreira se klasa `ImePrezime` koja nasleđuje klasu `Thread`. Klasa `ImePrezime` ima dva atributa: `rec` i `pauza`.

Konstruktor ove klase ima dva formalna argumenta čije vrednosti kopira u atribute. Reč `this` se odnosi na referencu na objekt klase `ImePrezime` čija je metoda pozvana kao što je u primeru to konstruktor. Ovim se omogućuje da se u naredbi dodele razreši konflikt imena šta se čemu dodeljuje. Tako je omogućeno da formalni argument ima isti naziv kao i pripadni atribut pa će korišćenjem reči `this` ime biti vezano za atribut datog objekta klase `ImePrezime`. Bez ovoga bi imena atributa i formalnog argumenta morala biti različita.

Metoda `run` klase `ImePrezime` u beskonačnoj petlji (`while(true)`) ispisuje za tekuću nit njen identifikator, naziv, prioritet i demonstvo. Tekuća nit je dostupna statičkom metodom `currentThread` klase `Thread`. Nabrojana svojstva tekuće niti se uzimaju metodama: `getID`, `getName`, `getPriority` i `isDeamon` respektivno. U programskoj liniji koja koristi metodu `isDeamon` testira se da li je tekuća nit demonska korišćenjem uslovnog operatora. Na kraju se u metodi run vrši suspendovanje niti čija se run metoda izvršava na period zadat parametrom pauza koji označava vreme u milisekudama. Nakon ove pauze nit nastavlja svoju beskonačnu while petlju.

U main metodi pokreće se garbage collector kao paralelena nit, a glavna programska nit kreira dve niti prosleđivanjem parametara za naziv niti i pauzu konstruktoru klase `ImePrezime`. Za svaku nit dodeljuje se prioritet (`setPriority`), naziv (`setName`) i demonstvo (`setDaemon`) te se niti pokreću. U primeru je dato da su obe niti demonske, tako da će nakon 5 sekundi pauze (`sleep(5000)`) main metoda dosegnuti kraj, a budući da su obe niti postavljene kao demonske (kao što je i garbage collector) program završava rad.

Sledi lista često korišćenih konstruktora klase Thread:

- `public Thread(Runnable target)`
    - konstruktor niti sa formalnim argumentom `Runnable` što znači da će stvarni argument biti referenca na klasu koja implementira interfejs `Runnable`
- `public Thread(Runnable target, String name)`

    - kao i prethodni konstruktor koji još specificira naziv niti kao drugi formalni argument
- `public Thread(ThreadGroup group, Runnable target)`
    - konstruktor niti u specificiranoj grupi niti
- `public Thread(ThreadGroup group, Runnable target, String name, long stackSize)`
    - isto kao i prethodni konstruktor uz specificiranje imena niti i veličine steka koji će se dodeliti niti

Sledi lista često korišćenih metoda klase `Thread`:

- `public final setName(String ime)`
    - postavljanje naziva niti
- `public final String getName()`
    - dohvatanje naziva niti
- `public long getId()`
    - dohvatanje jedinstvenog identifikatora niti:
- `boolean isAlive()`
    - provera da li je nit aktivna (da li se još uvek izvršava):
- `String toString()`
    - tekstualna reprezentacija naziva, prioriteta i naziva grupe kojoj nit pripada
- `public static Thread currentThread()`
    - kao i u primeru, dohvatanje objekta tekuće niti

3.1. Raspoređivanje niti

Kod raspoređivanja niti najvišeg prioriteta dobijaju najviše procesorskog vremena dok niti nižeg prioriteta dobijaju manje procesorskog vremena.

Nit je blokirana ako je uspavana ili izvršava funkciju čije je napredovanje blokirano. U slučaju blokiranja niti višeg prioriteta ostaje više procesorskog vremena nitima nižeg prioriteta.

Postavljanje prioriteta niti trebalo bi da bude usmereno ka cilju efikasnog izvršavanja. Izvršavanje algoritma ne sme biti zasnovano na prioritetima.

3.2. Metode sleep i yield

Daju se oblici metode `sleep` koja je u prethodnom primeru korišćena za suspendovanje (privremeno zaustavljanje izvršavanje niti) niti:

- `public static void sleep(long millis) throws InterruptedException`
    - uspavljivanje niti specificirani broj milisekundi
- `public static void sleep(long millis,int nanos) throws InterruptedException`
    - kao i prethodni sa dodatnom granulacijom u nanosekudama do 999999 nanosekundi

Metoda `yield` se koristi da tekuća nit omogući drugim nitima da dobiju procesor. Može se desiti da ista nit koja je pozvala ovu metodu opet dobije procesor. Metoda ima oblik: **`public static void`** `yield()`.

```
class PrikazReci extends Thread{
  static boolean radiYield;//oslobadjanje procesora
  String rec;            // rec koja se prikazuje
  static int n;          // koliko puta se rec prikazuje

  PrikazReci(String r){
    rec= r;
  }

  public void run(){
    for (int i=0; i< n; i++) {
      System.out.println(rec);
      if(radiYield) yield();}// šansa drugoj niti
  }

  public static void main(String[] arg){
    radiYield = new Boolean(arg[0]).booleanValue();
    n=Integer.parseInt(arg[1]);
    //za svaku rec po jedna nit
    Thread tekucaNit = currentThread();
    tekucaNit.setPriority(Thread.MAX_PRIORITY);
    for (int i=2; i<arg.length; i++)
      new PrikazReci(arg[i]).start();
  }
}

Ulazi i izlazi programa:
Poziv #1: java PrikazReci false 2 DA NE
Izlaz #1 (moguci izlaz):
DA
DA
NE
NE
Poziv #2: java PrikazReci true 2 DA NE
```

```
      Izlaz #2 (moguci izlaz):
      DA
      NE
      DA
      NE
```

Slika 3.5. Korišćenje metode yield

U primeru datom na slici 3.5 statički atribut `radiYield` se koristi kao logička promenljiva za pozivanje ili ne pozivanje `yield` metode. Ideja je da se pri pozivu aplikacije daju parametri korišćenja `yield` metode, broja iteracija ispisivanja reči i samih reči, respektivno.

Za svaku reč formira se po jedna nit koja: ispisuje tu reč, daje šansu drugoj niti ili ne pozivom ili ne metode yield i ponovo ispisuje reč zadati broj puta.

3.3. Završavanje niti

Povratkom iz svog metoda run nit završava svoje izvršavanje. Trebalo bi izbegavati zastarele metode eksplicitnog zaustavljanja niti: `stop` i `destroy`.

Prva metoda baca izuzetak `ThreadDeath`, hendler ukida nit pri čemu se otključavaju brave koje je nit zaključala. Druga metoda prekida nit bez otključavanja brava koje je nit zaključala.

Zaustavljanje niti bi trebalo izvesti tako da metoda run u petlji ispituje uslov kraja niti. Metodom `isAlive` proverava se da li se nit još izvršava.

3.4. Suspendovanje i reaktiviranje niti

Zastareli metodi za suspendovanje, reaktiviranje i zaustavljanje niti su: `suspend`, `resume` i `stop`, respektivno (slika 3.6).

```
public void interakcijaSaKorisnikom(){
  nit.suspend(); // suspenduje se nit cija je metoda
                 // interakcijaSaKorisnikom pozvana
  if (potvrdaSaTastature("Da/Ne? D/N"))
    nit.stop();          // za odabrano D
  else nit.resume();     // za odabrano N
}
```

Slika 3.6. Zastareli metod suspenzije i zaustavjanja niti

Metoda `interakcijaSaKorisnikom` suspenduje nit pozivom njene metode `suspend`, a onda metoda `potvrdaSaTastature` zahteva od korisnika da sa tastature unese odgovor da li zaista želi prekid niti. Unosom znaka D nit se prekida inače se nit reaktivira.

Videlo se da se nit može uspavati (suspendovati) pozivom metode sleep drugim rečima nit prelazi u blokirano stanje.

Metodi vezani za prekidanje niti su:

- `void interrupt()` – postavlja niti status prekida, a ako je nit u blokiranom stanju: ista se deblokira, nastaje izuzetak `InetrruptedException` i ne postavlja se status prekida
- `boolean interrupted()` – testira da li je tekuća nit bila prekinuta i poništava joj status prekida
- `boolean isInterrupted()` – testira da li je nit bila prekidana i ne menja status prekida

Metodama `sleep`, `wait` i `join` nit se dovodi u blokirano stanje. Ako se pozove metoda `join` neke niti čeka se da ta nit čija je metoda `join` pozvana završi i onda se nastavlja izvršavanje koda pozivajuće niti. Ovim je omogućeno da se ne pokupe "presni" rezultati koje generiše pozvana nit koji bi se koristili u pozivajućoj niti.

```
class Racunanje extends Thread {
  private double rez;
  public void run(){ rez = izračunaj(); }
  public double rezultat(){ return rez; }
  private double izračunaj() { ... }
}
class PrimerJoin{
  public static void main(String[] argumenti){
    Racunanje rnit = new Racunanje(); rnit.start();
    try{
      rnit.join();
      System.out.println("Rezultat je"+rnit.rezultat()) ;
    }
    catch(InterruptedException e){
      System.out.println("Prekid!");
    }
  }
}
```

Slika 3.7. Zastareli metod suspenzije i zaustavjanja niti

U primeru datom na slici 3.7. čeka se dok god nit `rnit` ne završi. Kada `rnit` završi, objekat te niti postoji i može mu se pristupiti i mogu se preuzeti rezultati.

Metoda `join` može imati sledeće parametre:

- `public final void join()`

```
throws InterruptedException
```

ova metoda čeka završetak niti

- ```
  public final void join(long millis)
  throws InterruptedException
  ```

metoda čeka završetak niti do maksimalno isteka zadatog vremena datog u milisekundama

- ```
  public final void join(long millis,int nanos)
  throws InterruptedException
  ```

metoda čeka završetak niti do maksimalno isteka zadatog vremena datog u milisekundama i dodatnog vremena u nanosekundama (opseg nanosekundi je 0 do 999999)

Posmatrajući stanja niti ista može biti:

- u stanju izvršavanja kada dobija procesor
- suspendovana, kada se njena aktivnost privremeno prekida, a nakon toga nastavlja izvršavanje odakle je bila prekinuta
- blokirana, kada čeka da pristupi deljenom resursu koga koristi druga nit
- završena.

3.5. Sinhronizacija niti

Komunikacija među nitima se realizuje deljenim resursom, odnosno, zajedničkim objektom.

Java virtuelna mašina i operativni sistem odlučuju o dodeli procesorskog vremena tako da postoji mogućnost da se jedna nit u toku pristupa deljenom objektu prekine i kontrola da drugoj niti koja pristupa ovom deljenom objektu i modifikuje njegovo stanje koje će izazvati grešku pri nastavku izvršavanja prve niti.

Ako nema sinhronizacije među nitima problem je da će modifikacija deljenih podataka (resursa) kojima niti paralelno pristupaju dati rezultat koji zavisi od redosleda pristupa ovih niti.

U sledećem primeru ilustruje se stanje kada dve niti istovremeno pristupaju deljenom resursu koji predstavlja račun na koji se vrši uplata prilikom prodaje goriva na dve benzinske pumpe (slika 3.8).

Nit 1	**Nit 2**
`novac=racun.uzmiStanje();`	
	`novac=racun.uzmiStanje();`

`novac+=cenaNatocenogGoriva;`	
	`novac+=cenaNatocenogGoriva;`
`racun.postaviStanje(zarada);`	
	`racun.postaviStanje(zarada);`

Slika 3.8. Dve niti nesinhronizovano pristupaju deljenom resursu

U ovom primeru na kraju obe uplate početni račun bi bio povećan samo za drugu uplatu. Ovakav program je nekorektan jer zavisi od slučajnog redosleda naredbi i spada u *Race Hazard* (neizvesna trka).

Ono što bi trebalo obezbediti je ekskluzivno pravo *get-modify-set* (uzmi vrednost, modifikuj je i postavi modifikovanu vrednost) sekvence što se postiže sinhronizacijom.

Potrebno je koristiti mehanizam zaključavanja objekata koji obezbeđuje da najviše jedna nit može da pristupa deljenom objektu u nekom periodu vremena.

Ovaj mehanizam je u Javi implementiran pomoću sinhronizovanih *(synchronized)* metoda i sinhronizovanih blokova.

Sinhronizacija niti zasnovana na bravi (lock) omogućuje da samo jedna nit može da pristupi sinhronizovanom objektu u jednom trenutku. Kada nit pristupi sinhronizovanom objektu ona zaključa taj objekat tako da mu osim nje ostale niti ne mogu pristupiti.

Mehanizam zaključavanja pozvane metode obezbeđen je modifikatorom metode synchronized tako da kada nit pozove sinhronizovan metod objekta ona dobija ključ brave objekta dok će druga nit koja pozove sinhronizovan metod istog objekta biti blokirana. Blokirana nit će biti deblokirana kada nit koja ju je blokirala napusti sinhronizovani metod.

Ovim mehanizmom je obezbeđeno uzajamno isključivanje niti nad deljenim objektom. Kada se nit nađe unutar sinhronizovane metode, nijedna druga nit ne može da pozove bilo koju sinhronizovanu metodu istog objekta.

Besmisleno je sinhronizovati konstruktor jer je u pitanju kreiranje objekta koji još ne postoji tako da mu druga nit ne može pristupiti.

```
class Racun{
  private double stanje;
  public Racun(double pocetniDepozit){
   stanje= pocetniDepozit;
  }
  public synchronized double uzmiStanje() {
    return stanje;
```

```
  }
  public synchronized void postaviStanje(double iznos){
    stanje+= iznos;
  }
}
```

Slika 3.9. Sinhronizacija računa

Na slici 3.9. dat je primer sinhronizacije računa gde je izbegnut problem *get-modify-set* sekvence (metode `uzmiStanje` i `postaviStanje` su sinhronizovane).

U slučaju sinhronizacije statičke metode klase zaključava se brava klase. Ovim mehanizmom je obezbeđeno uzajamno isključivanje niti nad celom klasom.

Brava klase nema uticaj na objekte te klase što znači da jedna nit može izvršavati sinhronizovani statički metod, a druga nit sinhronizovani nestatički metod.

Prilikom nasleđivanja klase redefinisanjem sinhronizovanog metoda u izvedenoj klasi ne nasleđuje se sinhronizovanost pri čemu ostaje sinhronizovano ponašanje metoda osnovne klase. Ovim će poziv sinhronizovanog metoda osnovne klase iz nesinhronizovanog metoda izvedene klase zaključati objekat dok se pozvani metod ne izvrši.

Suprotan slučaj je da postoji klasa sa nesinhronizovanim metodama, a potrebno je da one budu sinhronizovane. Rešenja su:

- izvođenje nove klase sa redefinisanim sinhronizovanim metodama koje pozivaju metode super klase
- korišćenje sinhronizovane naredbe za pristup objektu.

Sinhronizovana naredba ima oblik:

```
synchronized (izraz) { blok }
```

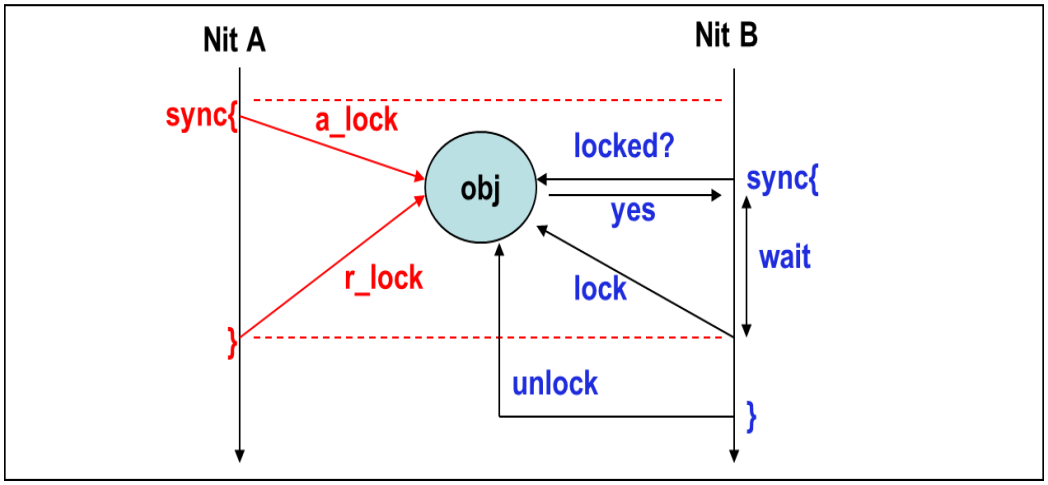
Sinhronizovana naredba zaključava objekat na koji upućuje rezultat izraza i izvršava se blok naredbi (slika 3.10). Ovim je omogućena finija granulacija sinhronizacije tako da se koriste sinhronizovane naredbe.

```
public static void kvadrat(int [] niz) {
  synchronized (niz) {
    for (int i=0; i<niz.length; i++){
      niz[i]*=niz[i];
    }
  }
}
```

Slika 3.10. Sinhronizovana naredba

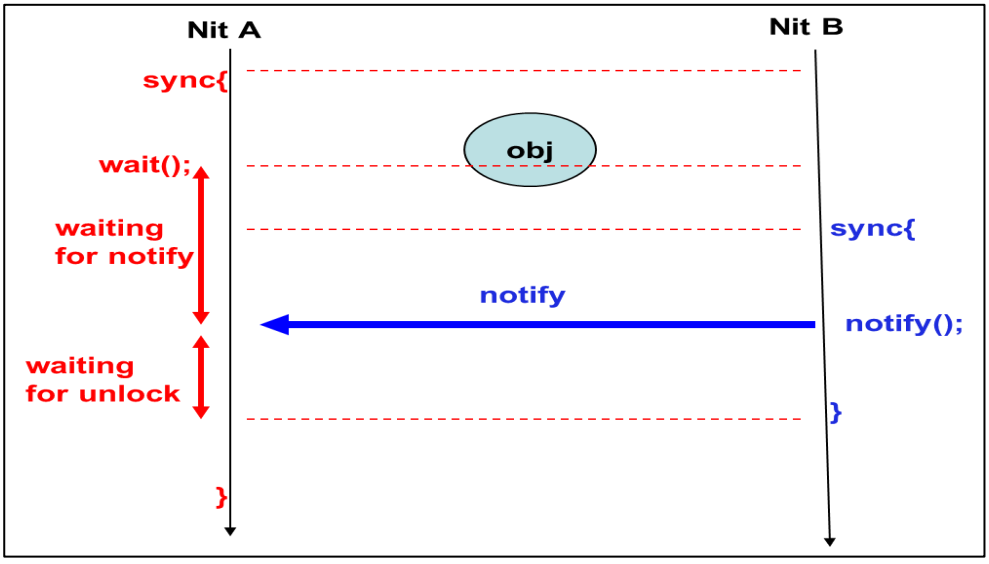
Na slici 3.11. dat je pristup deljenom objektu iz dve niti. U datom primeru nit `A` prva hronološki pristupa sinhronizovanom bloku i zaključava deljeni objekat (a_lock, *acquire lock*).

Nit `B` kasnije pokušava da pristupi sinhronizovanom bloku deljenog objekta i biva blokirana dok nit `A` ne oslobodi objekat. Kada nit `A` izađe iz sinhronizovanog bloka oslobodiće deljeni objekat (rlock, *release lock*). Sada nit `B` ima pravo pristupa, ulazi u sinhronizovani blok i zaključava ovaj objekat. Kada nit `B` napušta sinhronizovani blok otključaće objekat.



Slika 3.11. Pristup deljenom objektu iz dve niti

Problem je kada nit unutar sinhronizovanog bloka dugo čeka na neki događaj pri čemu i dalje onemogućava druge niti da koriste deljeni objekat.



Slika 3.12. Mehanizam wait/notify

Rešenje ovog problema objašnjeno je na slici 3.12. Koristi se metoda `wait` koja pripada klasi `Object` čime je obezbeđena njena dostupnost u svim klasama.

Metoda `wait` pozvana u sinhronizovanom bloku oslobađa zauzeti objekat i blokira izvršavanje niti koja je pozvala ovu metodu u atomskoj (nedeljivoj) operaciji sve dok druga nit ne pozove metodu `notify` nad istim objektom unutar sinhronizovanog boka.

Metoda `notify` pripada klasi `Object` tako da je dostupna svim klasama. Ova metoda obaveštava nit koja je prva pozvala svoju metodu `wait` da može nastaviti svoje izvršavanje što se dešava kada nit koja je pozvala metodu `notify` izađe iz sinhronizovanog bloka. Sada prva nit ponovo zaključava objekat i nastavlja dalje izvršavanje programskog koda.

Za obaveštavanje svih niti koje su bile pozvale svoju wait metodu i čekaju na notifikaciju koristi se metoda `notifyAll`.

Nakon poziva metode `notifyAll` u sinhronizovanom bloku pri izlasku iz sinhronizovanog bloka sve niti koje su dobile notifikaciju konkurisaće za procesorsko vreme.

Ako bi se neka od metoda `wait`, `notify`, `notifyAll` pozvala van sinhronizovanog bloka desio bi se izuzetak `IllegalMonitorStateException`.

Primer niti koja čeka uslov:

```
synchronized void metoda1() {
  ...
  while (!condition) wait();
  ...
}
```

Primer niti koja postavlja uslov na istinitu vrednost i obaveštava nit koja čeka na ispunjenje uslova:

```
synchronized void metoda2() {
  ...
  condition = true;
  notify();
  ...
}
```

Na slici 3.13. dat je primer korišćenja metoda `wait` i `notify` u klasi koja realizuje red čekanja. Nit koja pokuša da uzme element iz praznog reda biće stavljena u stanje čekanja dok se ne stavi element u red. Stavljanje elementa u red poziva notifikaciju koja će omogućiti niti koja je u stanju čekanja na element da se probudi i uzme element.

```
class Red {
  Element glava, rep;
```

```
  public synchronized void stavi(Element p) {
    if (rep==null) glava=p;
    else           rep.sledeci=p;
    p.sledeci=null;
    rep=p;
    notify();
  }
  public synchronized Element uzmi(){
    try {
      while(glava==null) wait();// čekanje na element
    }
    catch (InterruptedException e) {
      return null;
    }
    Element p=glava; // pamćenje prvog elementa
    glava=glava.sledeci;// izbacivanje iz reda
    if (glava==null) rep=null; // ako je prazan red
    return p;
  }
}
```

Slika 3.13. Mehanizam wait/notify

Sledi lista `wait` i `notify` oblika:

- **`public final void`** `wait(long milis)` **`throws`** `InterruptedException`
  - čekanje do na `milis` milisekundi
  - ako je `milis=0` čeka se do notifikacije
- **`public final void`** `wait(long milis, int nanos)` **`throws`** `InterruptedException`
  - čekanje do na `milis` milisekundi i `nanos` (fina granulacija) nanosekundi u opsegu do 999999
- **`public final void`** `wait()` **`throws`** `InterruptedException`
  - isto kao wait(0)
- **`public final void`** `notify()`
  - notifikuje nit koja prva čeka na uslov
- **`public final void`** `notifyAll()`
  - notifikuje sve niti koje čekaju na uslov

3.6. Problem deadlock-a

Problem deadlock-a nastaje kada dva objekta sa sinhronizovanim metodama čekaju na bravu onog drugog objekta i dolazi do uzajamnog blokiranja.

Primer: Objekat `A` iz svog sinhronizovanog metoda poziva sinhronizovan metod objekta `B` pri čemu objekat `B` iz svog sinhronizovanog metoda poziva sinhronizovan metod objekta `A`.

Na programeru je odgovornost da izbegne uzajamno blokiranje pošto Java nema mehanizme za detekciju i sprečavanje deadlock-a.

3.7. Sinhronizacija proizvođača i potrošača

U sledećem primeru pokazuje se sihronizovana realizacija proizvođača i potrošača. Deljeni resurs je proizvod (objekat klase `Q`). Vidi sliku 3.14.

```
class PrimerSinhroProizvodjacPotrosac {

  public static void main(String args[]) {
    Q q = new Q();
    new Producer(q);
    new Consumer(q);
  }
}
```

Slika 3.14. Deljeni resurs Q proizvođača i potrošača

```
class Producer implements Runnable {
  Q q;

  Producer(Q q) {
    this.q = q;
    new Thread(this, "Producer").start();
  }

  public void run() {
    int i = 0;
    while (true) {
      q.put(i++);
      try {
        Thread.sleep(500);
      }
      catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }
```

```
}
```

Slika 3.15. Nit proizvođača

Na slici 3.15. dat je kod proizvođača koji predstavlja nit koja proizvodi proizvod `q` tipa `Q` i poziva metodu `q.put` kojoj prosleđuje redni broj proizvoda `(i)`, a onda pravi pauzu od 500ms.

```
class Consumer implements Runnable {
  Q q;
  Consumer(Q q) {
    this.q = q;
    new Thread(this, "Consumer").start();
  }
  public void run() {
    while (true) {
      q.get();
      try {
        Thread.sleep(1000);
      }
      catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }
}
```

Slika 3.16. Nit potrošača

Na slici 3.16. dat je kod potrošača koji predstavlja nit koja troši proizvedeni proizvod `q` tipa `Q` i poziva metodu `q.get()` čime se proizvod potroši, a onda pravi pauzu od 1000ms.

```
class Q {
  int n;
  boolean valueSet = false;
  synchronized int get() {
    if (!valueSet)
    try {
      wait();
    }
    catch (InterruptedException e) {
      System.out.println("InterruptedException caught");
    }
    System.out.println("Got: " + n);
    valueSet = false;
```

```
    notify();
    return n;
  }
  synchronized void put(int n) {
    if (valueSet)
    try {
      wait();
    }
    catch (InterruptedException e) {
      System.out.println("InterruptedException caught");
    }
    this.n = n;
    valueSet = true;
    System.out.println("Put: " + n);
    notify();
  }
}

Izlaz:
Put: 0
Got: 0
Put: 1
Got: 1
Put: 2
Got: 2
...
```

Slika 3.17. Proizvod Q

U primeru datom na slici 3.17. potrošač čeka da proizvođač proizvede proizvod da bi ga konzumirao (metoda get), a proizvođač čeka da potrošač kupi proizvod da bi krenuo u proizvodnju sledećeg proizvoda (metoda put). Za oba slučaja nit koja čeka na nešto od navedenog ide u stanje čekanja sve dok ne bude notifikacija o stvaranju uslova za njeno dalje izvršavanje.

Još jedan primer sinhronizacije gde je deljeni resurs ping-pong tabla. Igrač čeka na potez (udarac) drugog igrača (Aca i Caca su kolega i koleginica sa mojih studija, prim.aut).

```
public class PingPongSto {
    private String sledeciIgrac = "";
    public synchronized boolean hit(String protivnik) {
       if (protivnik.compareTo("KRAJ") == 0) {
          sledeciIgrac = protivnik;
          notify();
          return false;
```

```
       }
      if(sledeciIgrac.compareTo("KRAJ")==0){
            notify();
            return false;
      }
      if (sledeciIgrac == "") {
        sledeciIgrac = protivnik;
        return true;
      }
      String tekuciigrac = Thread.currentThread().getName();
      if (tekuciigrac.compareTo(sledeciIgrac) == 0) {
            System.out.println("igra: "+ tekuciigrac );
            sledeciIgrac = protivnik;
            notify();
      }
      else {
         try {
            System.out.println("      "+ tekuciigrac +
            " ceka da "+sledeciIgrac+" igra.");
            wait();
         }
         catch (InterruptedException e) {
            System.out.println( e.getMessage());
         }
      }
      return true;
   }
}
```

Slika 3.18. PingPongSto kao deljeni resurs

```
public class Player implements Runnable {
     PingPongSto pingpongsto;
     String protivnik;
     int pauza;
     public Player(String protivnik,
                   PingPongSto pingpongsto, int pauza) {
        this.pingpongsto = pingpongsto;
        this.protivnik   = protivnik;
        this.pauza       = pauza;
     }
     public void run() {
       while (pingpongsto.hit(protivnik)) {
         try {
```

```
                Thread.sleep(pauza);
            }
            catch (InterruptedException e) {
              e.printStackTrace();
            }
        }
    }
}
```

Slika 3.19. Igrači ping-ponga realizovani kao niti

```
public class Game {
  public static void main(String args[]) {
      PingPongSto pingpongsto = new PingPongSto();
      Thread aca  = new Thread(new Player("Caca",
            pingpongsto, 500)); //oponenta, tabla, pauza
      Thread caca = new Thread(new Player("Aca" ,
            pingpongsto, 2000)); //oponenta, tabla, pauza
      aca.setName("Aca");
      caca.setName("Caca");
      aca.start();
      caca.start();
      try {
        Thread.sleep(6000);
      }
      catch (InterruptedException e) {
        e.printStackTrace();
      }
      pingpongsto.hit("KRAJ"); //zaustavi igrace
      System.out.println("Kraj igranja.");
     }
  }
```

Slika 3.20. Glavna klasa za igranje ping-ponga

```
Izlaz:
igra: Caca
igra: Aca
      Aca ceka da Caca igra.
igra: Caca
igra: Aca
      Aca ceka da Caca igra.
igra: Caca
igra: Aca
      Aca ceka da Caca igra.
Kraj igranja.
```

Slika 3.21. Izlaz aplikacije Game

U datom primeru Aca i Caca igraju stoni tenis. Zajednički resurs je ping-pong sto (tabla) na kom igraju. Igra se 6000ms.

Igraču je dodeljena nit koja ima naziv koji odgovara imenu igrača (slika 3.20). Kreiranjem igrača prosleđuju se parametri: ime oponenta, ping-pong sto na kom igraju (zajednički) i pauza koja modeluje vreme potrebno za igrač vrati lopticu oponentu. Aca je brži tako da će morati da čeka na Cacin udarac. Kada se tabli prosledi KRAJ igra prestaje.

PingPongSto (slika 3.18) je deljeni resurs koji ima sinhronizovanu metodu `hit`. U ovoj metodi se ispisuje koji igrač (nit) je igrao i na čiji se udarac čeka (slika 3.21). Igrači (slika 3.19) udaraju lopticu (poziv metode `hit`) i igra traje dok se tabli iz glavne niti (`main`) ne prosledi KRAJ kada se završavaju metode `run` igrača.

3.8. Modifikator volatile

Ako u višenitnim programima dve ili više niti dele istu promenljivu, pri čemu svaka nit može da zadrži svoju (lokalnu) kopiju zajedničke promenljive, može se dogoditi da se original (master) promenljive ažurira u različitim vremenima, čime može doći do razlike lokalne kopije i originala.

Modifikator `volatile` sprečava kompajler da primeni bilo kakvu optimizaciju koda koja se odnosi na promenljivu označenu kao volatile. Ovaj modifikator garantuje globalni redosled čitanja i pisanja volatile promenljive. Ovim je rešeno da bilo koja nit koja pristupa ovakvoj promenljivoj čita njenu tekuću (stvarnu) vrednost, a ne njenu keširanu vrednost.

`Volatile` obezbeđuje da dato polje neće biti keširano i različite niti će videti ažurirano polje. U primeru na slici 3.22. kontrola završetka rada niti iz glavne metode će raditi pravilno.

```
public class VolatileTest extends Thread{
    volatile boolean keepRunning = true; //probajte bez volatile
    public void run() {
        long count=0;
        while (keepRunning) {
            count++;
        }
        System.out.println("Thread terminated."+count);
    }
    public static void main(String[] args) throws
InterruptedException {
        VolatileTest t = new VolatileTest();
        t.start();
        Thread.sleep(1000);
        t.keepRunning = false;
```

```
        System.out.println("keepRunning set to false.");
    }
}
IZLAZ
keepRunning set to false.
      Thread terminated.1763087972
```

Slika 3.22. Volatile

Rezime poglavlja konkurentno programiranje

U poglavlju konkurentno programiranje obrađeno je: kreiranje niti, raspoređivanje niti, korišćenje metoda sleep i yield, završavanje niti, suspendovanje i reaktiviranje niti, sinhronizacija niti, problem deadlocka, sinhronizacija proizvođača i potrošača i korišćenje modifikatora volatile.

Zadaci za proveru znanja iz poglavlja konkurentno programiranje:

1. Neka sporija (zla) nit umanjuje ocenu, a (dobra) brža nit uvećava ocenu. Neka ocena kreće od 7. Svaka promena ocene se ispisuje u konzoli, a program se zaustavlja kada se dosegne ocena 10.
2. Neka je dato pet filozofa koji jedu štapićima za okruglim stolom. Štapića ima 5. Da bi filozof jeo mora da uzme svoj levi i svoj desni štapić. Napisati konkurentni program koji omogućuje da se filozofi "lepo" najedu. Ispisivati statistiku koja se odnosi na broj zalogaja filozofa dok ne istekne zadato vreme jela.
3. Napisati program koji realizuje račun u banci. Niti su klijenti koji imaju mogućnost da uplaćuju i podižu novac sa računa.
4. Napisati konkuretni program koji ima tri proizvođača i pet potrošača. Proizvođači proizvode kolače koji moraju da se konzumiraju u redosledu kojim su napravljeni bez obzira koji je potrošač došao na red da ih konzumira.
5. Napisati program gde 2 niti menjaju slučajno dati niz od 12 slova. Prva nit menja slučajno slovo u prvoj polovini niza, dok druga nit menja slovo u drugoj polovini niza. Prva nit je brža od druge niti. Cilj je da se dobije reč POPOKATEPETL. Kada nit pogodi dato ciljno slovo onda ga više ne menja. Ispisivati sve promene sve dok se ne dobije POPOKATEPETL.

4. GENERIČKI TIPOVI

Cilj poglavlja je da student ovlada korišćenjem generičkih tipova za kreiranje generičkih klasa, generičkih kolekcija, generičkih statičkih metoda, korisničkih generičkih kolekcija te upotrebe generičkih tipova pri nasleđivanju.

Kao i u programskom jeziku C++ i u programskom jeziku Java su implementirani šabloni (generici, generički tipovi) koji služe za generalizaciju koda. Osnovna ideja je da se kod napiše šablonski, tako da navođenje poziva takvog koda za dati konkretni tip koji "menja" šablonski opšti tip. Ovim se dobija kao da smo sami napisali takav kod za navedeni tip, čime je izbegnuto pisanje po funkcionalnosti istog programskog koda za različite tipove. Budući da se radi o "istom" kodu ponašanje je isto samo se radi o različitom tipu. Sa druge strane, korišćenjem generika, postavljena je kontrola tipa referenci koje sadrži npr. neka kolekcija.Posmatra se primer:

```
List list = new ArrayList();
list.add(new Integer(2));
list.add("a String");
```

u kome se u listu `list` dodaju reference na objekte različitog tipa. U primeru `list` sadrži referencu na objekat tipa `Integer` i referencu na objekat tipa `String`. Ovo zahteva kastovanje povratne vrednosti koja se dobija pozivom metode `list.get(indeks)`:

```
Integer integer = (Integer) list.get(0);
String  string  = (String)  list.get(1);
```

4.1. Generičke kolekcije

Java generici omogućuju da se postavi tip koji kolekcija može da prihvati čime dalje nije potrebno kastovanje uzete vrednosti iz kolekcije:

```
List<String> strings = new ArrayList<String>();
```

ili drugi način pisanja sa *diamond* operatorom:

```
List<String> strings = new ArrayList<>();
```

dodaje se referenca na string:

```
strings.add("a String");
...
```

i nije potrebno kastovanje povratne vrednosti metode `get` jer je reč o listi koja sadrži reference na objekte klase `String`:

```
String aString = strings.get(0);
```

trivijalni ispis elemenata liste:

korišćenjem for petlje:

```
for(String aString : strings){
  System.out.println(aString);
}
```

korišćenjem iteratora:

```
Iterator<String> iterator = strings.iterator();
while(iterator.hasNext()){
    System.out.println(iterator.next());
}
```

Sledi primer za generički skup i pripadne ispise korišćenjem for petlje te iteratora:

```
Set<String> set = new HashSet<String>();
String string1 = "a string";
set.add(string1);
...
for(String aString : set){
   System.out.println(aString);
}
Iterator<String> iterator = set.iterator();
while(iterator.hasNext()){
   System.out.println(iterator.next());
}
}
```

Generička mapa `map` služi za mapiranje reference na `Integer` ka referenci na `String`:

```
Map<Integer, String> map = new HashMap<Integer, String>();
Integer key1   = new Integer(123);
String  value1 = "value 1";
map.put(key1, value1);
...
String value1_1 = map.get(key1);
```

trivijalni ispis elemenata mape:

ključeva i vrednosti korišćenjem iteratora po ključevima:

```
Iterator<Integer> keyIterator = map.keySet().iterator();
while(keyIterator.hasNext()){
  Integer aKey   = keyIterator.next();
  String  aValue = map.get(aKey);
  System.out.println("Key - value "+ aKey +" - "+ aValue);
}
```

samo vrednosti korišćenjem iteratora po vrednostima:

```
Iterator<String>  valueIterator = map.values().iterator();
```

```
while(valueIterator.hasNext()){
  System.out.println(valueIterator.next());
}
```

ključeva i vrednosti korišćenjem for petlje po kolekciji ključeva:

```
for(Integer aKey : map.keySet()) {
    String aValue = map.get(aKey);
    System.out.println("" + aKey + ":" + aValue);
}
```

samo vrednosti korišćenjem for petlje po vrednostima:

```
for(String aValue : map.values()) {
    System.out.println(aValue);
}
```

4.2. Generičke klase

Ideja je da se za isto ponašanje objekata različitog tipa omogući pisanje šablona klase ili metode gde se navodi simbol za tip (npr. `T`), a onda se konkretizacija tipa navodi prilikom poziva (konstruktora, metode). U toku prevođenja se vrši provera o doslednosti poziva koji se odnosi na generike.

Kod generičke klase potrebno je pre tela klase navesti generik. Neka je data generička klasa `GenericFactory<T>` koja se kreira za tip `T`. Klasa ima atribut `theClass` koji je referenca na objekat tipa `Class` i inicijalno je postavljen na `null`.

```
public class GenericFactory<T> {
   Class theClass = null;
   public GenericFactory(Class theClass) {
            this.theClass = theClass;
   }
   public T createInstance()
     throws IllegalAccessException, InstantiationException {
              return (T) this.theClass.newInstance();
   }
}
```

Konstruktor ove klase prihvata referencu na objekat tipa `Class` koja se dodeljuje atributu `theClass`. Metoda `createInstance()` ima povratnu vrednost `T` i može da generiše izuzetke `IllegalAccessException` i `InstantiationException`. U telu ove metode vraća se referenca na kreirani objekat klase `T`. Primer poziva za klasu `GenericFactory<T>`:

```
GenericFactory<MyClass> factory =
          new GenericFactory<MyClass>(MyClass.class);
```

```
MyClass myClassInstance = factory.createInstance();
```

Posmatra se trivijalan primer generičke klase `Box<T>` (slika 4.1) koja sadrži `put` i `get` metode koje respektivno postavljaju referencu na objekat tipa `<T>`, odnosno, vraćaju istu, respektivno. U main metodi deklarisane su i definisane dve reference: `Box<Integer> integerBox` i `Box<String> stringBox` koje referenciraju objekte datog tipa. Pozivaju se metode `put` i jednog i drugog objekta kojima se prosleđuje argument potrebnog tipa (`Integer`, `String`). Sledi poziv metoda za ispis na konzolu u kome se pozivaju metode `get` za objekte koji su referencirani referencama `integerBox` i `stringBox`.

```
public class Box<T> {
  private T t;
  public void put(T t) {
    this.t = t;
  }
  public T get() {
    return t;
  }
  public static void main(String[] args) {
    Box<Integer> integerBox = new Box<Integer>();
    Box<String>   stringBox = new Box<String>();
    integerBox.put(new Integer(10));
    stringBox.put(new String("Hello World"));
    System.out.printf("Integer Value :%d\n\n", integerBox.get());
    System.out.printf("String Value :%s\n", stringBox.get());
  }
}
```

Izlaz:

```
Integer Value :10
String Value :Hello World
```

Slika 4.1. Generička klasa Box<T>

Razmislite o dodavanju statičke generičke metode koja će ispisivati vrednost koju referencira `t`.

4.3. Generičke statičke metode

Generičke statičke metode moraju imati vlastitu generičku deklaraciju. Posmatra se primer statičke generičke metode `addAndReturn`:

```
public static <T> T addAndReturn(T element,
                                 Collection<T> collection){
    collection.add(element);
```

```
    return element;
}
```

Pre povratnog tipa generičke metoda postavlja se znak za generik `<T>`. U primeru povratna vrednost metode je `T`, a parametri su: `element` kao referenca na tip `T` i `collection` ako generička kolekcija referenci na tip `T`. Metoda dodaje `element` u `collection` i vraća element. Primeri poziva statičke generičke metode `addAndReturn` u klasi u kojoj je metoda definisana:

```
String stringElement = "stringElement";
List<String> stringList = new ArrayList<String>();
String theElement = addAndReturn(stringElement, stringList);

Integer integerElement = new Integer(123);
List<Integer> integerList = new ArrayList<Integer>();
Integer theElement = addAndReturn(integerElement, integerList);

String stringElement = "stringElement";
List<Object> objectList = new ArrayList<Object>();
Object theElement = addAndReturn(stringElement, objectList);
```

Neka je sada rešenje instanciranja objekta klase `T` rešeno pozivom statičke generičke metode `getInstance`:

```
public static <T> T getInstance(Class<T> theClass)
    throws IllegalAccessException, InstantiationException {
      return theClass.newInstance();
}
...
String string   = getInstance(String.class);
MyClass myClass = getInstance(MyClass.class);
```

Posmatra se primer statičke generičke metode `printArray` koja pripada klasi `GenericMethodTest` (slika 4.2). Ova metoda prihvata niz koji sadrži reference na objekte tipa `E`, a onda ispisuje elemente `E`. U `main` metodi kreirana su tri niza referenci na objekte tipa: `Integer`, `Double` i `Character`, respektivno. Nizovi su kontejnerski popunjeni, a onda je pozvana generička metoda za sva tri niza.

```
public class GenericMethodTest {
  public static <E> void printArray(E[] inputArray) {
    for (E element : inputArray) {
                        System.out.printf("%s ", element); }
    System.out.println();
  }
  public static void main(String args[]) {
    Integer[] intArray = { 1, 2, 3, 4, 5 };
    Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
```

```
    Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };
    System.out.println("Array integerArray contains:");
    printArray(intArray); // pass an Integer array
    System.out.println("\nArray doubleArray contains:");
    printArray(doubleArray); // pass a Double array
    System.out.println("\nArray characterArray contains:");
    printArray(charArray); // pass a Character array
  }
}
```

Izlaz:

```
Array integerArray contains:
1 2 3 4 5
Array doubleArray contains:
1.1 2.2 3.3 4.4
Array characterArray contains:
H E L L O
```

Slika 4.2. Statički generički metod

4.4. Korisnička generička kolekcija

Generici se mogu koristiti za kreiranje kolekcije. Kolekcija mora da implementira interfejs `Iterable` koji ima metode: `hasNext()`, `next()` i `remove()`. Za ovakav slučaj pripadni kod bi izgledao kao što sledi:

```
public class MyCollection<E> implements Iterable<E>{
    public Iterator<E> iterator() {
        return new MyIterator<E>();
    }
}

public class MyIterator <T> implements Iterator<T> {
    public boolean hasNext() {
      //implementacija
    }
    public T next() {
        //implementacija
    }
    public void remove() {
        //implementacija
    }
}

public static void main(String[] args) {
    MyCollection<String> stringCollection =
                                        new MyCollection<String>();
    for(String string : stringCollection){ ... }
```

```
}
```

4.5. Nasleđivanje i generički tipovi

Kod nasleđivanja i korišćenja generika koriste se sledeći slučajevi:

- tipovi su nepoznatog tipa
- tipovi su instanca date klase ili podklase
- tipovi su instanca klase ili superklase

Neka se koriste tri klase: klasa `A`, klasa `B` izvedena iz `A` i klasa `C` izvedena iz `A`.

```
public class A { ... }
public class B extends A { ... }
public class C extends A { ... }
```

Posmatra se lista koja sadrži reference na nepoznat tip (oznaka `<?>`) i koja je argument metode `processElements`. Ova metoda ispisuje elemente liste. Kreirana je referenca na objekat tipa `List<A>`, konkretno kao new `ArrayList<A>()` te je ista prosleđena metodi `processElements`.

```
public static void processElements(List<?> elements){
   for(Object o : elements){
      System.out.println(o);
   }
}
...
List<A> listA = new ArrayList<A>();
processElements(listA);
```

Neka je sada metoda `processElements` takva da prihvata listu koja sadrži reference na objekte potklase klase `A` (ili same klase `A`).

```
public static void processElements(
                                   List<? extends A> elements){
   for(A a : elements){
      System.out.println(a.getValue()); //u A je getValue()
   }
}
```

Sada su mogući pozivi:

```
List<A> listA = new ArrayList<A>();
processElements(listA);
List<B> listB = new ArrayList<B>();
processElements(listB);
List<C> listC = new ArrayList<C>();
processElements(listC);
```

Upotreba `<? extends>` je podesna ako se želi dohvatanje (GET) elementa „`A`".

Neka je sada metoda `processElements` takva da prihvata listu koja sadrži reference na objekte superklase klase `A` (ili same klase `A`).

```
public static void insertElements(List<? super A> list){
    list.add(new A());
    list.add(new B());
    list.add(new C());
}
```

Sada su mogući pozivi:

```
List<A> listA = new ArrayList<A>();
insertElements(listA);
List<Object> listObject = new ArrayList<Object>();
insertElements(listObject);
```

Upotreba `<? super A>` je podesna ako se želi dodavanje (PUT) elementa „`A`".

Statička generička metoda `maximum` u klasi `MaximumTest` (slika 4.3) ima generičku deklaraciju `<T extends Comparable<T>>` koja znači da je podržana komparacija tipa `T` preko `Comparable` interfejsa (preporučujem `<T extends Comparable<? super T>>`). U `main` metodi se u ispisu poziva metoda `maximum` kojoj se prosleđuju tri argumenta uporedivih tipova (celobrojni, pokretni zarez, string), a kao odgovor dobija maksimalna od prosleđenih vrednosti za dati tip.

```
public class MaximumTest{
   // najveci od tri
   public static <T extends Comparable<T>> T maximum(T x, T y, T
z) {
      T max = x;
      if ( y.compareTo( max ) > 0 ){ max = y; }
      if ( z.compareTo( max ) > 0 ){ max = z; }
      return max;
   }
   public static void main( String args[] ) {
      System.out.printf( "Max of %d, %d and %d is %d\n\n",
                  3, 4, 5, maximum( 3, 4, 5 ) );
      System.out.printf( "Maxm of %.1f,%.1f and %.1f is %.1f\n\n",
                  6.6, 8.8, 7.7, maximum( 6.6, 8.8, 7.7 ) );
      System.out.printf( "Max of %s, %s and %s is %s\n","pear",
        "apple", "orange", maximum( "pear", "apple", "orange" ) );
   }
}
```

Izlaz:

```
Max of 3, 4 and 5 is 5
Maxm of 6.6,8.8 and 7.7 is 8.8
Max of pear, apple and orange is pear
```

Slika 4.3. Klasa MaximumTest

Rezime poglavlja generički tipovi

U poglavlju generički tipovi obrađeno je kao što sledi: korišćenje generičkih tipova za kreiranje generičkih klasa, generičke kolekcije, generičke statičke metode, korisničke generičke kolekcije te upotreba generičkih tipova pri nasleđivanju.

Zadaci za proveru znanja iz poglavlja generički tipovi:

1. Napisati generički klasu koja realizuje stek (*stack*) različitih tipova podataka.
2. Napisati generičku klasu koja realizuje red (*queue*) za različite tipove podataka.
3. Napisati program za loto izvlačenje koji koristi generički definisanu klasu Hashtable.
4. Napisati generičku klasu MyCollection koja implementira interfejs Iterable.
5. Napisati generičku klasu Par<U,V> koja realizuje par objekata.

5. GRAFIČKI KORISNIČKI INTERFEJS - GUI

U prethodnim poglavljima korišćene su konzolne aplikacije kod kojih se kao tekstualni izlaz koristi konzola. Cilj poglavlja je da student ovlada programiranjem grafičkih interfejsa korišćenjem tehnologija: AWT (*Abstract Windowing Toolkit*), Swing i JavaFX.

Grafički korisnički interfejs GUI (Graphical User Interface) predstavlja interakciju korisnika i programa preko grafičkih elemenata.

U ovom poglavlju će biti predstavljena tri tehnologije grafičkog korisničkog interfejsa koje se koriste u Java programskom jeziku:

- AWT Abstract Windowing Toolkit
- Swing
- JavaFX

5.1. AWT

Početak rada AWT GUI aplikacije polazi od metode main kao i u tradicionalnim programima.

AWT GUI je baziran na upravljanju događajima (*event-driven*) koje generišu grafičke komponente prozora tako da aplikacija reaguje na događaje obrađujući ih u posebnim metodama koje predstavljaju obrađivače događaja.

Aplikacija se izvršava u trenucima kada nastupe događaji koje ista može da obrađuje.

Podrška za programiranje aplikacija baziranih na AWT smeštena je u paketu java.awt. Reč apstraktni alati znači da ne zavise od platforme na kojoj će se izvršavati.

Paket java.awt sadrži klase i interfejse koji podržavaju pogramiranje samostalnih aplikacija baziranih na GIU interfejsu.

U literaturi se sreće pojam grafičkih komponenti pod nazivima kontrole ili vidžiti (widgets). Primeri vidžita su: ekranski tasteri (button), polja za tekst (text box), radio-dugmad (radio-button), polja potvrde (checkbox) itd.

Klasa Component je zajednička klasa za sve GUI kontrole, naravno, ako se izuzme klasa Object kao implicitna osnovna klasa svih klasa.

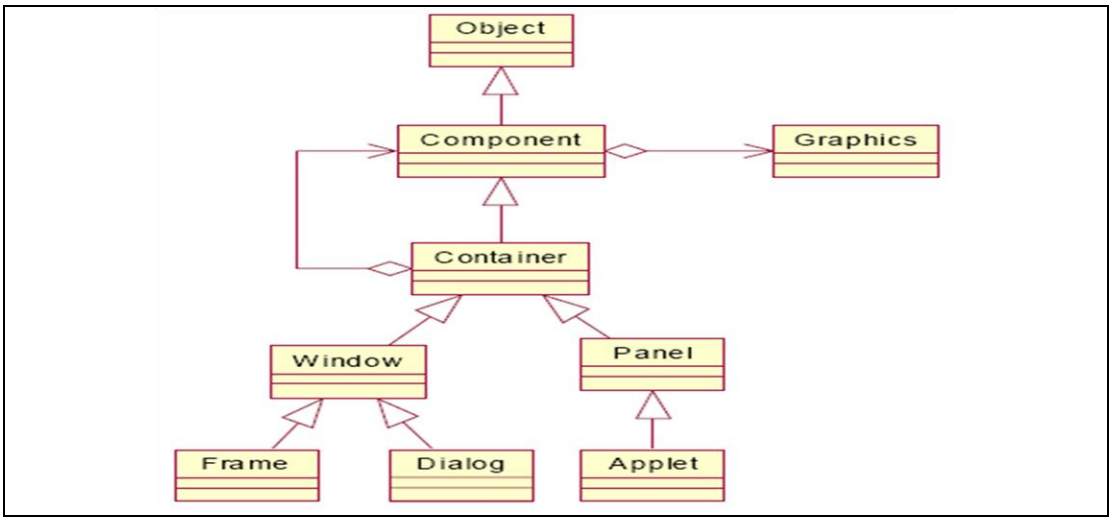
Na slici 5.1. dat je UML dijagram klasa vezanih za klasu Component.

Klasa Component odnosi se na grafički element koji:

- ima poziciju;

- ima veličinu;
- može se iscrtati na ekranu;
- prihvata ulazne događaje.

Klasa komponent ima navigaciju po refrenci ka klasi `Graphics`. Objekat se dohvata metodom `getGraphics()`.



Slika 5.1. UML Dijagram klasa vezanih za klasu Component

Klasa `Graphics` obezbeđuje pisanje i crtanje po komponentama. Grafičke operacije modifikuju bitove bit mape unutar odsecajućeg regiona pri čemu se koriste tekuće vrednosti boje, fonta...

Klasa `Container` nasleđuje klasu `Component` pri čemu `Container` sadrži druge AWT komponente. Omogućuje dodavanje, dohvatanje, prikazivanje, prebrojavanje i uklanjanje komponenata, prosleđivanje događaja svojim komponentama te raspoređivanje komponenti.

Klase `Window` i `Panel` nasleđuju klasu `Container`.

Klasa `Window` odnosi se na prozore najvišeg nivoa i obuhvata metode za rad sa prozorima.

Klasa `Panel` je generički kontejner koji može sadržavati i druge panele.

Klase `Frame` nasleđuje klasu `Window` i u noj se kreira glavni prozor aplikacije. Ovaj prozor može da sadrži traku menija i naslov.

Klasa `Dialog` nasleđuje klasu `Window` i služi za kreiranje prozora dijaloga. Dijalog nema meni i mora da ima roditeljski prozor.

Klasa `java.applet.Applet` je osnovna klasa za sve aplete i izvedena je iz klase `Panel`.

5.1.1. GUI komponente

Sledi spisak nekih klasa GUI komponenata sa kratkim opisom:

- `Canvas` - kanvas (pravougaoni prostor na ekranu po kojem se može crtati)
- `Label`
  - koristi se za prikazivanje natpisa koji se ne menja interaktivno već programski
  - metode ove klase upisuju, čitaju i poravnavaju natpis
- `Button`
  - programsko dugme (ekranski taster) označeno natpisom
- `Checkbox` - služi kao polje za potvrdu ili radio-dugme
- `CheckboxGroup` – grupiše polja za potvrdu koja se tad ponašaju kao radio dugmad (samo jedno dugme u grupi može biti potvrđeno)
- `List` - lista
- `Choice` - padajuća lista (*combo-box*)
- `TextField` - tekstualno polje
- `TextArea` - prostor za višelinijski tekst
- `ScrollBar` - klizač
- `ScrollPane` - panel sa klizačima za pomeranje sadržane komponente

5.1.2. Podela AWT događaja

Podela AWT događaja izvršena je u dve grupe:

- događaji niskog nivoa koji se odnose na prikaz elementarnih zbivanja u sistemu prozora ili elementarne ulaze
  - događaji komponenti, kontejnera, fokusa i prozora
    - događaji komponenata se generišu pri promeni pozicije, veličine i vidljivosti
    - događaji kontejnera se generišu kada se komponenta dodaje ili uklanja iz kontejnera
    - događaji fokusa se generišu kada komponenta dobija ili gubi fokus (unos sa) tastature
    - događaji prozora daju informaciju o bazičnom stanju prozora
  - elementarni događaji miša ili tastature
- semantički događaje koji su rezultat korisničkih akcija koje su specifične

za komponente

- događaje akcije generišu
  - ekranski tasteri (pritisak), stavke menija, liste i tekst polja
- događaji prilagođenja se generišu kada korisnik promeni vrednost klizača (*scrollbar*)
- članske događaje generiše izbor jedne od stavki iz liste
- tekstualni događaji se generišu kada se menja tekst u polju za tekst ili prostoru za tekst

U prilozima, u tabelama A1.a. i A1.b. dati su događaji koje generišu AWT komponente.

5.1.3. Upravljanje događajima

Upravljanje događajima (event-driven) koji pokreću GUI aplikaciju rešeno je u Javi na dva načina:

- centralizovano upravljanje događajima
- obrada događaja pomoću izvora i osluškivača

Centralizovano upravljanje događajima je zastareo koncept. Ideja je da postoji kompletna lista statičkih celobrojnih konstanti, gde konstanta pripada tačno jednom događaju. Kada nastupi događaj kreira se klasa `Event` koja se prosledi metodi `handleEvent`. Sada se proverava atribut id objekta događaja poređenjem sa odgovarajućom konstantom koja se odnosi na ispitivani događaj.

Drugim rečima objekat komponente koja generiše događaj obrađuje događaj, a centralni metod (`handleEvent`) obrade koji je zadužen za sve događaje rešen je kao razgranata kontrolna struktura što nije u duhu objektno orijentisanog programiranja.

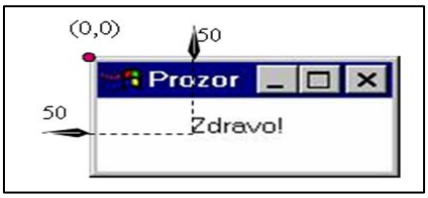
Na primer pritisak programskog dugmeta:

- postaviće vrednost atributa `id` klase `Event` na vrednost `ACTION_EVENT`
- postaviće se atribut target klase `Event` na `Button` koji je izazvao događaj
- atribut `arg` klase `Event` postavlja se na vrednost natpisa pritisnutog ekranskog tastera

```
import java.awt.*;
public class Prozor extends Frame {
  public Prozor() {
      super("Prozor");
      setSize(180,80);//ili setSize(new Dimension(180,80));
      setVisible (true);
```

```
  }
  public void paint(Graphics g) {
      g.drawString("Zdravo!",50,50);
  }
  public boolean handleEvent(Event e) {
      if(e.id==Event.WINDOW_DESTROY){  System.exit(0);}
      else return false;
  }
  public static void main(String args[]){
      Prozor prozor = new Prozor();
}
```

Slika 5.2. Centralizovano upravljanje događajima



Slika 5.3. Izgled aplikacije Prozor

U primeru datom na slici 5.2. klasa Prozor nasleđuje klasu Frame. Metode klase su: konstruktor klase, paint, handleEvent i main.

Konstruktor klase `Prozor`:

- realizovan je bez formalnih argumenata.
- poziva konstruktor super klase (`Frame`) i prosleđuje mu naslov.
- poziva metodu `setSize` klase `Component` prosleđujući parametre koji označavaju dimenzije prozora.
- poziva metodu `setVisible(boolean)` klase `Window` i dozvoljava ili sprečava prikazivanje prozora.

Metod `paint`:

- ima formalni argument koji je refrenca na objekat klase `Graphics` koji se koristi da se ažurira prikaz komponente u njegovom podrazumevanom kanvasu (površini za crtanje)
- pripada klasi `Component` i automatski se poziva iz AWT niti za inicijalno iscrtavanje te prilikom promene veličine ili pomeranja ili pokrivanja prozora.
- metod iscrtava objekat
- u primeru ispisuje tekst "`Zdravo!`" u prozoru aplikacije, počevši od tačke (50,50) što je prikazano na slici 5.3. Ishodište koordinatnog sistema

grafičkog prikaza je gornji levi ugao komponente. Odsecajući (clipping) region grafičkog prikaza je pravougaonik oko komponente

Metod `handleEvent`:

- metod klase `Component` koji se automatski poziva kada se desi događaj unutar komponente
- ako metod `handleEvent` uspešno obradi događaj vraća se `true`
- ako metod ne obradi događaj uspešno, vraća se false te se događaj prosleđuje roditeljskoj komponenti u hijerarhiji objekata
- u datom primeru ovaj metod detektuje zatvaranje prozora aplikacije kao događaj označen statičkom konstantom WINDOW_DESTROY

Metod `main`:

- kreira objekat klase `Prozor` koji je i tipa `Frame`
- startuje se AWT nit, kao i nit `garbage collector`-a
- čeka na završetak AWT niti

5.1.4. Obrada događaja pomoću izvora i osluškivača

Od Java verzije 1.1 uvoden je novi koncept događaja i obrade događaja. Koncept je baziran na izvorima (*sources*) koji generišu događaje i osluškivačima (*listeners*) događaja koji obrađuju događaje. Koriste se biblioteke `java.awt` i `java.awt.event`.

Klase osluškivača obavezno implementiraju interfejs jednog ili više osluškivača događaja.

Potrebno je jedan ili više objekata osluškivača događaja registrovati kod nekog izvora događaja. Sada će registrovani osluškivači biti obaveštavani od izvora o događajima koje ovaj izvor generiše.

Metode obrade događaja, hendlere (*handlers*) određuju interfejsi odgovarajućeg osluškivača.

U literaturi se sreću pojmovi prosleđivanje i delegiranje za ovaj način obrade događaja. Pojmovi se odnose na delegiranje prava obrade događaja svakom objektu koji implementira interfejs odgovarajućeg osluškivača.

Kada se desi događaj kreira se objekat odgovarajuće klase (koja nasleđuje klasu `Event`) koji opisuje ovaj događaj, a potom se taj objekat prosleđuje registrovanom osluškivaču događaja.

Sada je obrada događaja za razliku od centralizovane obrade događaja strogo objektno orijentisana.

Na primer sada se kod pritiska programskog dugmeta dešava sledeće:

- događaj se evidentira se u objektu klase `ActionEvent`
- događaj se obrađuje u implementiranoj metodi `actionPerformed` klase koja implementira interfejs `ActionListener` pri čemu je prosleđeni argument ove metode referenca na objekat klase `ActionEvent` u kom je evidentiran događaj
- izvor događaja `e.getSource()` je objekat tipa `Button` koji je izazvao događaj
- metoda `getActionCommand` klase `ActionEvent` vraća akcionu komandu dodeljenu ekranskom tasteru koji je izazvao događaj.

5.1.5. Osluškivači događaja

Da bi se realizovao koncept obrade događaja baziran na izvoru i osluškivačima događaja u programu se moraju programirati sledeća 3 elementa:

- u klasi osluškivača implementirati interfejs nekog osluškivača ili naslediti klasu koja implementira interfejs osluškivača

  primer:

  ```
  class Obradjivac implements ActionListener {
  class Obradjivac extends WindowAdapter {
  ```

  ovde je klasa `WindowAdapter` primer klase koja implementira interfejs WindowListener i to tako da su sve metode ovog interfejsa nadjačane sa praznim kodom (objašnjenje ovog postupka sledi u poglavlju o adapterima).
- kod izvora događaja registrovati instancu klase osluškivača događaja

  primer:

  ```
  izvordogadjaja.addActionListener( obradjivac );
  ```
- implementirati sve metode koje propisuje interfejs osluškivača

  primer:

  ```
  public void actionPerformed(ActionEvent e){/*...*/}
  ```

  u datom primeru interfejs `ActionListener` ima samo jednu metodu `actionPerformed`, da ima više metoda sve bi morale biti implementirane.

Osluškivač može biti realizovan u:

- klasi u kojoj je komponenta koja izaziva događaj
- unutrašnjoj klasi (prozorskoj ili neimenovanoj)
- drugoj klasi

U prvom slučaju registrovanje osluškivača događaja ima oblik:

```
dugmeA.addActionListener(this);
```

Kod neimenovanih klasa definiše se referenca na objekat klase koja implementira interfejs. Ako se želi koristiti referenca na klasu koja implementira dati interfejs koriste se sledeća realizacija:

```
ActionListener actlis = (new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
      System.exit(0);
    }
});
```

sada je poziv oblika:

```
dugmeA.addActionListener(actlis);
dugmeB.addActionListener(actlis);
```

pri čemu je identično ponašanje oba dugmeta.

U slučaju da se u programu ne koristi referenca na klasu koja implementira dati interfejs realizacija je sledeća:

```
dugmekraj.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent ev) {
            System.exit(0);
     }
});
```

Ako realizacija osluškivača nije u klasi kojoj pripada komponenta, onda je poziv oblika:

```
dugmeA.addActionListener(new KlasaOsluskivaca());
```

gde je `KlasaOsluskivaca` neka druga klasa u koja implementira interfejs datog osluškivača. Ovde je trivijalno navedeno da klasa nema formalnih argumenata, što je stvar programera i njegove realizacije ove klase.

Sledi primer delegirane obrade događaja gde se registruje klik miša na programskom dugmetu (button-u). Videti sliku 5.4.

```
import java.awt.*; import java.awt.event.*;
public class Dogadjaj
extends Frame implements ActionListener {
      private Button dugme;
      private int broj=0;
      public Dogadjaj(){
            super("Dogadjaj"); setSize(200,200);
            kreirajDugme();
            setVisible(true);
      }
      private void kreirajDugme(){
```

```
            dugme = new Button("Pritisni");
            add(dugme);
            dugme.addActionListener(this);
      }
      public void actionPerformed(ActionEvent e) {
            dugme.setLabel("Pritisnuto "+ ++broj +". put");
      }
      public static void main(String[] args){
            Dogadjaj d = new Dogadjaj();
      }
}
Izlaz:
```

Slika 5.4. Obrada klika na programsko dugme

U datom primeru instanca `dugme` je izvor događaja kome je delegiran osluškivač `ActionListener` implementiran u klasi `Dogadjaj` te stvarni argument metoda `addActionListener` ima vrednost `this`. U klasi `Dogadjaj` implementirana je metoda `actionPerformed` u kojoj labela dugmeta služi za prikaz koliko puta je dugme pritisnuto.

U prilozima u tabeli A2. dati su najčešće korišćeni događaji i odgovarajući osluškivači.

5.1.6. Klase adaptera

U prethodnom primeru korišćen je interfejs `ActionListener` koji ima samo jednu metodu što ne predstavlja problem pri implementaciji. Međutim, ako interfejs ima dosta metoda sve se moraju implementirati. Osnovni problem nije sama prazna implementacija jer današnja razvojna okruženja mogu da generišu potreban kod već je problem u smanjenju čitljivosti koda. Neka interfejs ima 10 metoda i neka je u programu potrebno koristiti samo jedan metod, posledica je da kod postaje glomazan, nečitak i teži za održavanje.

Rešenje problema je korišćenje klase adaptera. U AWT-u je ugrađena klasa adapter za svaki osluškivač koji ima više od jednog metoda. Adapter implementira sve metode interfejsa osluškivača kao prazne metode.

Nedostatak korišćenja adaptera je što klasa koja nasleđuje adapter ne može istovremeno naslediti i drugu klasu.

Adapter se koristi na sledeći način:

- iz odgovarajućeg adaptera se izvodi klasa željenog osluškivača
- izvedena klasa redefiniše samo željene metode obrade događaja
- umesto implementacije interfejsa radi se nasleđivanje

Slede dva primera u kojima se daje primer korišćenja adaptera interfejsa `WindowListener`. Prvi primer (slika 5.5) koristi unutrašnju klasu koja nasleđuje adapter, a drugi primer pokazuje korišćenje anonimne klase koja nasleđuje adapter (slika 5.6). U oba primera obezbeđena je funkcionalnost zatvaranja prozora aplikacije.

```
import java.awt.*;
import java.awt.event.*;
public class DogadjajProzora extends Frame {
      class AdapterProzora extends WindowAdapter{
            public void windowClosing(WindowEvent we){
                  System.exit(0);
            }
      }
      public DogadjajProzora() {
            super("Dogadjaj Prozora");
            setSize(280,80);
            addWindowListener(new AdapterProzora());
            setVisible(true);
      }
      public void paint(Graphics g) {
            g.drawString("Zdravo!",50,50);
      }
      public static void main(String args[]){
            DogadjajProzora p = new DogadjajProzora();
      }
}
```

Slika 5.5. Unutrašnja klasa nasleđuje WindowAdapter

```
//primer resenja pomocu anonimne klase
import java.awt.*;
import java.awt.event.*;
public class DogadjajProzora extends Frame {
      public DogadjajProzora() {
```

```
            super("Anonimni Adapter");
            setSize(280,80);
            addWindowListener(new WindowAdapter(){
                  public void windowClosing(WindowEvent we){
                        System.exit(0);
                  }
            });
            setVisible(true);
      }
      public void paint(Graphics g) {
            g.drawString("Zdravo!",50,50);
      }
      public static void main(String args[]){
            DogadjajProzora p = new DogadjajProzora();
      }
}
```

Slika 5.6. Anonimni adapter

U oba primera korišćena je klasa adaptera gde je redefinisana samo metoda `windowClosing` tako da se klikom na ikonu za zatvaranje prozora izlazi iz aplikacije pozivom metode `System.exit(0)`.

U prilozima u tabelama A3.a. i A3.b. dati su osluškivači, pripadni adapteri i metode.

5.1.7. Obrada događaja računarskog miša

Interfejsi za osluškivanje događaja računarskog miša su kao što sledi:

- `MouseListener` koji ima 5 metoda:
  - `public void mousePressed (MouseEvent me);`

    događaj: pritisnuto dugme
  - `public void mouseClicked (MouseEvent me);`

    događaj: pritisnuto i otpusteno dugme na istoj poziciji gde je pritisnuto
  - `public void mouseReleased (MouseEvent me);`

    događaj: otpusteno dugme na različitoj poziciji od pozicije gde je pritisnuto
  - `public void mouseEntered (MouseEvent me);`

    događaj: kursor usao u polje komponente
  - `public void mouseExited  (MouseEvent me);`

događaj: kursor izasao iz polja ko komponente

- `MouseMotionListener` koji ima 2 metode koje se odnose na kretanje miša:
  - `public void mouseMoved (MouseEvent me);`

    miš je pomeren bez pritiskanja dugmeta
  - `public void mouseDragged (MouseEvent me);`

    miš je pomeren sa pritisnutim dugmetom
- `MouseWheelListener` koji ima metodu koja se odnosi na pomeranje točkića miša:
  - `public void mouseWheelMoved(MouseWheelEvent e);`

Klasa `MouseEvent` pripada paketu java.awt.event i ima sledeći konstruktor:

- ```
  public MouseEvent(Component source, int id,
                    long when, int modifiers,
                    intx,int y,
                    int clickCount,
                    boolean popupTrigger )
  ```
  - `source` : komponenta koja je izazvala događaj
  - `id` : tip događaja (npr. `MOUSE_CLICKED`, `MOUSE_DRAGGED`, ...)
  - `when` : timestamp trenutak kada se događaj desio
  - `modifiers` : modifikatori koji određuju da li je pritisnut `<ALT>, <SHIFT>, <MOUSE_BUTTONx>`
  - `x,y` : koordinate tačke gde se nalazio pointer pri nastanku događaja
  - `clickCount` : broj "klikova" kojima je izazvan događaj
  - `popupTrigger`: informacija da li je događaj izazvao pojavu pop-up menija

```
import java.awt.*;
import java.awt.event.*;
public class OsluskivacMisa
            extends Frame
            implements MouseListener {
  private String t="... ceka se na dogadjaj misa ...";
```
```

```
  public OsluskivacMisa(){
      super("Osluskivac misa");
      setSize(300,100);
      addMouseListener(this);
      setVisible(true);
  }
  public void paint(Graphics g){
      g.drawString(t,50,50);
  }
  public void mouseClicked (MouseEvent d){
     t="Dogadjaj: clicked";repaint();
  }
  public void mouseEntered (MouseEvent d){
     t="Dogadjaj: entered";repaint();
  }
  public void mouseExited  (MouseEvent d){
     t="Dogadjaj: exited"; repaint();
  }
  public void mousePressed (MouseEvent d){
     t="Dogadjaj: pressed";repaint();
  }
  public void mouseReleased(MouseEvent d){
     t="Dogadjaj: released";repaint();
  }
  public static void main(String[] args){
     OsluskivacMisa d=new OsluskivacMisa();
  }
}
```

Slika 5.7. Osluškivač miša

U primeru datom na slici 5.7. klasa `OsluskivacMisa` nasleđuje klasu `Frame` i implementira interfejs `MouseListener`. Svih 5 metoda ovog interfejsa se moraju implementirati.

Metoda `paint` ispisuje koji se od 5 događaja koji se odnose na događaj koji generiše miš dogodio.

U konstruktoru je navedena linija koda `addMouseListener(this)` kojom se nalaže da navedenih 5 događaja miša obrađuje osluškivač implementiran u samoj klasi (odatle `this`).

Metoda `repaint` se koristi da se prozor ponovo iscrta. Ovaj metod poziva metod `paint`. Metod `paint` se poziva kada se koriste metodi: `setVisible`(true), `show` (zastareo metod), `repaint` i `update`. `Repaint` je asinhrona metoda koja poziva

metodu update, a ova metodu paint.

U primeru datom na slici 5.8. demonstrira se korišćenje interfejsa MouseWheelListener. Pomeranjem točkića miša menja se vrednost brojača koja se prikazuje na koordinatama 50,50 datog frejma.

```
import java.awt.*;
import java.awt.event.*;

public class MouseWheelTest extends Frame {
    int brojac;

    public MouseWheelTest() {
        super();
        brojac = 0;
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent we){
            System.exit(0);
       }});
       MouseWheelListener listener =
                           new MouseWheelListener() {
            int colorCounter;
            private static final int UP = 1;
            private static final int DOWN = -1;

            public void mouseWheelMoved(MouseWheelEvent e) {
              int count = e.getWheelRotation();
              int direction = (count <= 0)?UP:DOWN;
              brojac+=direction;
              repaint();
            }
        };
        addMouseWheelListener(listener);
    }

    public void paint(Graphics g){
        g.drawString( ""+brojac, 50,50);
    }

    public static void main(String args[]) {
        Frame frame = new MouseWheelTest();
        frame.setSize(300, 300);
        frame.setVisible(true);
    }
}
```

Izlaz (pomeranje točkića 1 korak dole, a onda dva gore):

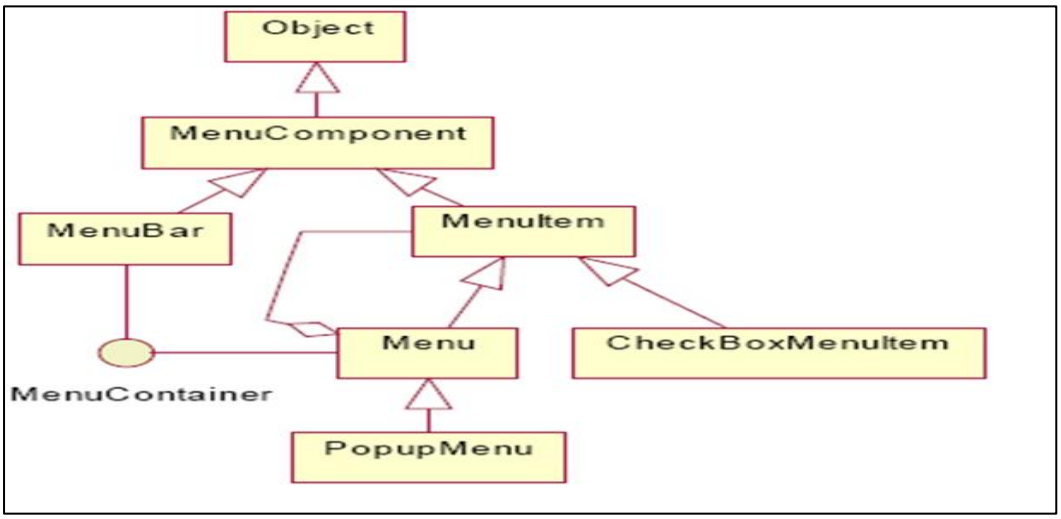
Slika 5.8. Osluškivač pomeranja točkića miša

Startovanjem programa prikazuje se početna vrednost brojača (vrednost 0), a onda ako se pomeri točkić miša jedan pomeraj dole vrednost brojača postaje -1, a onda dva pomeraja gore pri čemu vrednost brojača postaje 1.

5.1.8. Meniji

UML dijagram klasa menija dat je na slici 5.9. Objekat klase `MenuBar` sadrži jedan ili više objekata klase `Menu`. Klasa `MenuBar` nasleđuje klasu `MenuComponent` i odnosi se na traku menija koja se pridružuje prozoru aplikacije. Postavljanje trake menija u `Frame` obavlja se metodom `setMenuBar` klase `Frame`.

Padajuće menije realizuje klasa `Menu` koja nasleđuje klasu `MenuItem`. Kaskadni meniji omogućeni su time što objekat klase `Menu` može sadržati druge `Menu` objekte. Pored stavki menija u meni je moguće dodavati i separatore.



Slika 5.9. UML dijagram klasa menija

Klasa `MenuItem` realizije pojedinačne stavke menija i nasleđuje klasu `MenuComponent`. Omogućuje postavljanje i čitanje labele stavke menija.

Klasa `MenuComponent` je bazna klasa koja sadrži metode za rad sa menijima.

`CheckboxMenuItem` realizuje stavke menija koje mogu biti obeležene potvrdom. Ova klasa nasleđuje klasu `MenuItem`. Pripadne metode postavljaju znak potvrde i očitavaju status potvrđenosti.

Posebna klasa koje se odnosi na iskačuće (*popup*) menije jeste očekivano `PopupMenu` klasa. Ova klasa nasleđuje klasu `Menu`. Ovo su često korišćeni meniji gde se pritiskom suprotnog tastera miša dobija lista stavki ovog menija.

Interfejs `MenuContainer` sadži metode koje moraju implementirati klase koje sadrže objekte vezane za menije. Klase `Menu`, `MenuBar` i implementiraju interfejs

MenuContainer.

Obrada događaja iz menija bazirana je na interfejsu ActionListener tako da klasa koja hvata događaje iz menija (odabiranje stavke) mora da implementira ovaj interfejs (njegovu metodu actionPerformed). Ova klasa se registruje kao osluškivač objekta klase Menu korišćenjem metode addActionListener. Metodom getActionCommand klase ActionEvent dohvata se akciona komanda koja je podrazumevano naziv stavke menija. Akciona komanda se može promeniti metodom setActionCommand(String) klase MenuItem.

```
import java.awt.*;
import java.awt.event.*;

public class PrimerMenija
             extends Frame
             implements ActionListener {

  String izborIzMenija = "Izaberite stavku iz menija...";

  public PrimerMenija() {
        super("Primer Menija");
        setSize(300,200);
        dodajMenije();
        setVisible(true);
  }

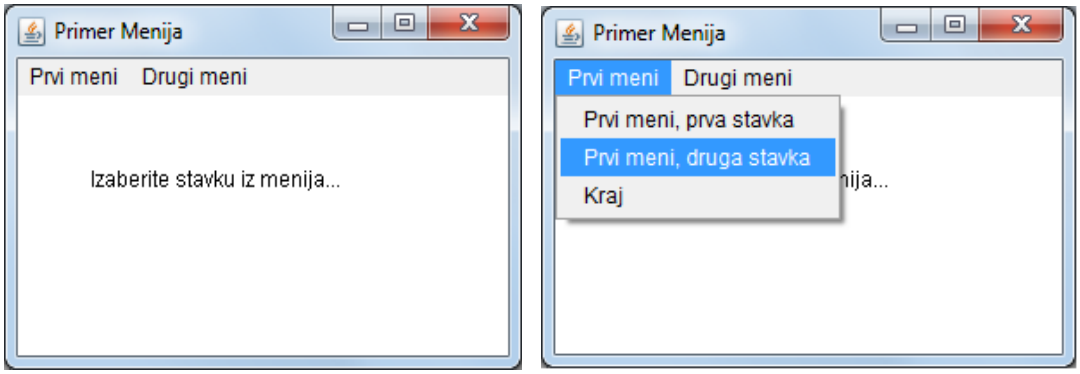
  void dodajMenije() {
      MenuBar trakaMenija = new MenuBar();
      Menu prviMeni = new Menu("Prvi meni");
      Menu drugiMeni = new Menu("Drugi meni");
      prviMeni.add("Prvi meni, prva stavka");
      prviMeni.add("Prvi meni, druga stavka");
      prviMeni.add("Kraj");
      prviMeni.addActionListener(this);
      drugiMeni.add("Drugi meni, prva stavka");
      drugiMeni.add("Drugi meni, druga stavka");
      drugiMeni.addActionListener(this);
      trakaMenija.add(prviMeni);
      trakaMenija.add(drugiMeni);
      setMenuBar(trakaMenija);
  }

  public void paint(Graphics g) {
            g.drawString(izborIzMenija,50,100);
  }

  public void actionPerformed (ActionEvent e) {
      String komanda=e.getActionCommand();
      if(komanda.equals("Kraj"))
            System.exit(0);
```

```
        else{
              izborIzMenija = "Izabrali ste "+komanda+".";
              repaint();
        }
    }
    public static void main(String args[]){
        PrimerMenija prozor = new PrimerMenija();
    }
}
```

Slika 5.10. Kreiranje menija



Slika 5.11. Izgled menija

U primeru datom na slici 5.10. klasa `PrimerMenija` u metodi `dodajMenije` kreira meni aplikaciju kao što sledi:

- kreira se objekat klase MenuBar
- kreiraju se objekti klase Menu
- u objekte klase Menu dodaju se stavke menija korišćenjem metode add(String)
- kreirani objekti klase Menu se registruju kod osluškivača `ActionListener` koji je implementiran u klasi `PrimerMenija`
- u kreirani objekat klase MenuBar dodaju se kreirani objekti klase Menu korišćenjem metoda add(Menu)
- na kraju se montira traka menija MenuBar na Frame pozivom metode setMenuBar(MenuBar)
- u metodi `actionPerformed` koristi se metoda `getActionCommand` klase `ActionEvent` koja vraća string komande (stavke) menija, a zatim vrši poređenje metodom `equals` da bi se uradilo zatvaranje aplikacije ili da se ispiše (metodom `repaint` koja poziva metodu `paint`) u kanvasu koja je stavka menija bila odabrana.

Izgled implementiranih menija prikazan je na slici 5.11.

Stara tehnika obrade događaja iz menija zasnivala se na prepoznavanju

ACTION_EVENT događaja, tako da se u centralnoj metodi handleEvent testiralo da li je generisani događaj ACTION_EVENT, a onda da li je komponenta koja je generisala događaj bila MenuItem. Koristi se testiranje da li je atribut target objekata Event instanca MenuItem klase korišćenjem metode instanceof.

Sledeći primer dat na slici 5.12. ilustruje korišćenje: klase CheckboxMenuItem, enumeracije, kaskadnog menija te interfejsa ActionListener i ItemListener koji su implementirani u istoj klasi:

```
import java.awt.*;
import java.awt.event.*;

public class PrimerMenija2
             extends Frame
             implements ActionListener, ItemListener{

  private String izbor = "Izaberite stavku menija...";
  private CheckboxMenuItem cbmi;

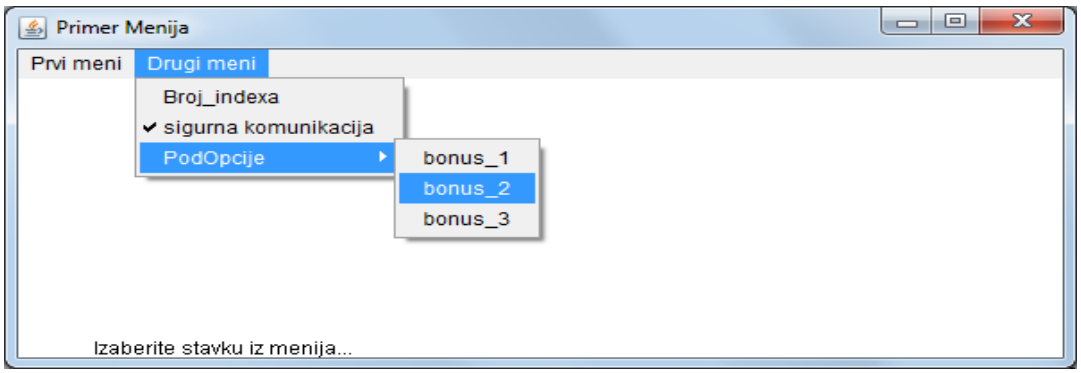
  public enum MyMenu{ //malo vezbanja enumeracije
    IME,PREZIME,KRAJ,BROJ_INDEXA,BONUS_1,BONUS_2,BONUS_3;
  }

  public PrimerMenija2() {
      super("Primer Menija");
      setSize(600,250);
      addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent we){
                  System.exit(0);
            }
      });
      dodajMenije();
      setVisible(true);
  }

  void dodajMenije() {
      MenuBar trakaMenija = new MenuBar();
      Menu prviMeni = new Menu("Prvi meni");
      Menu drugiMeni = new Menu("Drugi meni");
      prviMeni.add("Ime");
      prviMeni.add("Prezime");
      prviMeni.add("Kraj");
      prviMeni.addActionListener(this);
      drugiMeni.add("Broj_indexa");
      cbmi=new CheckboxMenuItem("sigurna komunikacija");
      cbmi.setState(true);
      cbmi.addItemListener(this);
      drugiMeni.add(cbmi);
      Menu podmeni = new Menu("PodOpcije");
      podmeni.add("bonus_1");
```

```
        podmeni.add("bonus_2");
        podmeni.add(new MenuItem("bonus_3"));
        podmeni.addActionListener(this);
        drugiMeni.add(podmeni);
        drugiMeni.addActionListener(this);
        trakaMenija.add(prviMeni);
        trakaMenija.add(drugiMeni);
        setMenuBar(trakaMenija);
    }
    public void paint(Graphics g) {
        g.drawString(izbor,50,250);
    }
    public void actionPerformed (ActionEvent e) {
        String komanda=e.getActionCommand();
        switch(MyMenu.valueOf(komanda.toUpperCase())){
          case IME          : izbor = "IME"; break;
          case PREZIME      : izbor = "PREZIME"; break;
          case KRAJ         : System.exit(0); break;
          case BROJ_INDEXA  : izbor = "BROJ INDEXA"; break;
          default           : izbor = komanda;
              }
              repaint();
        }
        public void itemStateChanged(ItemEvent e) {
          izbor = "sigurna komunikacija = " + cbmi.getState();
          repaint();
        }
        public static void main(String args[]){
              PrimerMenija2 prozor = new PrimerMenija2();
        }
}
```

Slika 5.12. Meni sa CheckboxMenuItem elementom



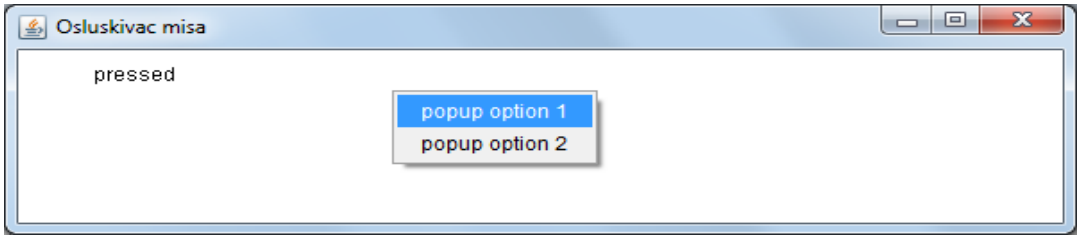
Slika 5.13. Izgled aplikacije PrimerMenu2

U ovom primeru upotrebljena je enumeracija `MyMenu` u kojoj su navedene simboličke konstante koje će biti korišćene u ispitivanju koja je stavka menija odabrana. U metodi `actionPerformed` interfejsa `ActionListener` uzima se akciona komanda (metodom `getActionCommand`), a onda se ista konvertuje u velika slova (metodom `toUpperCase`) i ispituje u selektoru. Na osnovu odabrane stavke izlazi se iz aplikacije ili se ispisuje izabrana stavka u kanvasu. Ako je odabrana stavka klase `CheckboxMenuItem`, onda je obrada događaja u metodi `itemStateChanged` interfejsa `ItemListener`, gde se metodom `getState` klase `itemStateChanged` proverava da li je ovakva stavka čekirana ili ne. Na slici 5.13. dat je grafički izgled opisanog primera.

```
import java.awt.*;
import java.awt.event.*;
public class OsluskivacMisa
             extends Frame
             implements MouseListener, ActionListener {
  private PopupMenu popup = new PopupMenu();
  private String t="Ceka se na dogadjaj ...";
  public OsluskivacMisa(){
      super("Osluskivac misa");
      setSize(300,100);
      MenuItem mi = new MenuItem("popup option 1");
      popup.add(mi);
      mi = new MenuItem("popup option 2");
      popup.add(mi);
      popup.addActionListener(this);
      add(popup);
      addMouseListener(this);
      setVisible(true);
  }
  public void paint(Graphics g){
      g.drawString(t,50,50);
  }
  public void mouseClicked (MouseEvent d){
      if (d.getButton()==MouseEvent.BUTTON3)
            popup.show(this, d.getX(), d.getY());
      t="clicked"; repaint();
  }
  public void mouseEntered (MouseEvent d){
     t="entered"; repaint();
  }
  public void mouseExited  (MouseEvent d){
     t="exited"; repaint();
```

```
  }
  public void mousePressed (MouseEvent d){
     t="pressed"; repaint();
  }
  public void mouseReleased(MouseEvent d){
     t="released";repaint();
  }
  public void actionPerformed(ActionEvent ae){
     if(ae.getActionCommand().equalsIgnoreCase("popup option 2"))
      System.exit(0);
  }
  public static void main(String[] args){
      OsluskivacMisa d=new OsluskivacMisa();
  }
}
```

Slika 5.14. Primer iskačućeg menija



Slika 5.15. Izgled iskačućeg menija

U primeru datom na slici 5.14. implementirani su intefejsi `MouseListener` i `ActionListener` koji su potrebni za događaje klik mišem suprotnim tasterom i odabiranje stavke iz padajućeg menija respektivno. Dodate su dve stavke u iskačući meni (`popup option 1` i `popup option 2`) kome je dodat osluškivač `ActionListener`.

U metodi `mouseClicked` ispituje se da li je događaj potekao od suprotnog tastera miša i ako jeste aktivira se iskačući meni na poziciji miša gde je nastupio ovaj događaj. Metoda `equalsIgnoreCase` klase String vrši poređenje dva stringa pri čemu se ignoriše veličina znakova. Izgled iskačućeg manija dat je na slici 5.15.

5.1.9. Dijalozi

Postoje dva tipa dijaloga koji se mogu kreirati korišćenjem klase `Dialog`:

- modalni dijalozi
    - kada se otvori modalni dijalog fokus se ne može preneti na druge prozore aplikacije
- nemodalni dijalozi

- kada se otvori nemodalni dijalog fokus se može prenositi i na druge prozore aplikacije i naravno ponovo vraćati na nemodalni dijalog

Konstruktor klase `Dialog` ima oblik:

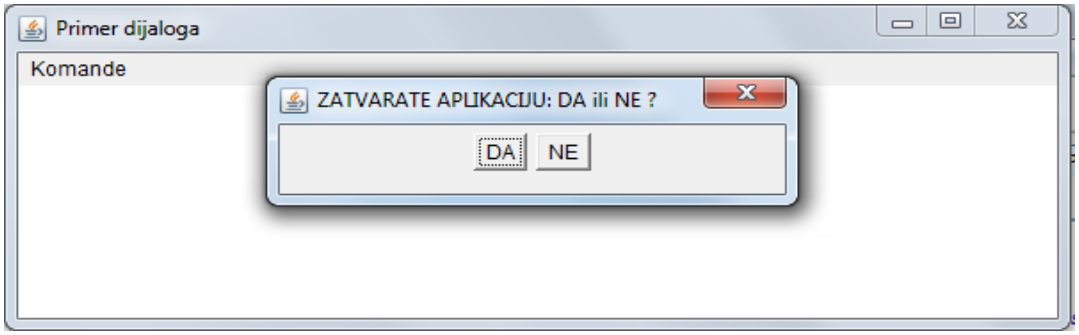
```
public Dialog (Frame   roditelj,
               String  naslov,
               boolean modalni)
```

Na slici 5.16. dat je dijalog čiji je roditelj glavni prozor aplikacije, naslov je Dijalog i dijalog je nemodalni. Klasa Dijalog realizovana je kao unutrašnja klasa klase `PrimerDijaloga`.

```
import java.awt.*;
import java.awt.event.*;
public class PrimerDijaloga
             extends Frame
             implements ActionListener{
  class Dijalog extends Dialog {
      private Button da = new Button("DA");
      private Button ne = new Button("NE");
      Dijalog(Frame roditelj) {
            super(roditelj,
                  "ZATVARATE APLIKACIJU: DA ili NE ?",
                  false);
            setSize(300,80);
            addWindowListener(new WindowAdapter(){
                  public void windowClosing(WindowEvent we){
                        setVisible(false);
                  }
            });
            this.setLayout(new FlowLayout());
            da.setActionCommand("Zatvori aplikaciju");
            add(da);
            da.addActionListener((PrimerDijaloga)roditelj);
            add(ne);
            ne.addActionListener((PrimerDijaloga)roditelj);
      }
  }
  private Dijalog dijalog;
  public PrimerDijaloga() {
      super("Primer dijaloga"); setSize(400,400);
      dodajMenije();
      dijalog = new Dijalog(this);
      setVisible(true);
  }
```

```
  void dodajMenije() {
      MenuBar trakaMenija = new MenuBar();
      Menu meni = new Menu("Komande");
      meni.add("Otvori");
      meni.add("Zatvori");
      meni.add("Kraj");
      meni.addActionListener(this);
      trakaMenija.add(meni);
      setMenuBar(trakaMenija);
  }
  public void actionPerformed (ActionEvent e) {
      String komanda=e.getActionCommand();
      if ( (komanda.equals("Zatvori aplikaciju")) ||
           (komanda.equals("Kraj")              )     )
           System.exit(0);
       else
            if ( (komanda.equals("NE")) ||
                 (komanda.equals("Zatvori")))
               dijalog.setVisible(false);
            else dijalog.setVisible(true);
  }
  public static void main(String args[]){
      PrimerDijaloga prozor = new PrimerDijaloga();
  }
}
```

Slika 5.16. Korišćenje dijaloga



Slika 5.17. Izgled aplikacije PrimerDijaloga

Nakon kreiranja dijaloga, isti se može otvarati i zatvarati pozivom metode `setVisible(Boolean)`. U unutrašnjoj klasi dijalog ima dodat osluškivač `WindowListener` realizovan preko `WindowAdapter`-a u kome je redefinisana metoda `windowClosing` da dijalog učini nevidljivim.

Aktiviranjem linija menija dijalog se prikazuje na ekranu, uklanja sa ekrana ili

se izlazi iz aplikacije. Sam dijalog reaguje na odabiranje programskih dugmadi (DA i NE) kojima se određuje da li se zatvara dijalog ili aplikacija.

U kodu je postavljeno da je osluškivač ovih događaja `ActionListener` u roditeljskoj klasi `PrimerDijaloga`. Metoda `setLayout` koristi se za postavljanje raspoređivača komponenti o čemu će biti detaljniji opis u materijalu koji sledi. Na slici 5.17. dat je izgled aplkacije `PrimerDijaloga`.

5.1.10. Paneli

Klasa Panel je kontejnerska klasa koja predstavlja površinu na koju se smeštaju komponente. Ova klasa nasleđuje klasu Container. Na panel se mogu smeštati drugi paneli.

Klasa može da reaguje na događaje čiji su izvori: miš, tastatura i promena fokusa.

Paneli se definišu, postavlja im se pozadina, dodaju im se komponente (npr. ekranski tasteri)

```
import java.awt.*;
import java.awt.event.*;
public class PrimerPanela
             extends Frame
             implements ActionListener{
  private Label labela =
                 new Label("panel cija se nijansa menja");
  private Button kraj =
          new Button("Zatvori aplikaciju");
  private Button menjajtekst =
          new Button("Menjanje nijanse");
  private Panel gornjipanel = new Panel();
  private Panel donjipanel  = new Panel();
  public PrimerPanela(String naziv) {
      super(naziv);
      setSize(300,100);
      dodajPanele();
      addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e){
                  dispose();
            }
      });
      kraj.setActionCommand("KRAJ");
      kraj.addActionListener(this);
      menjajtekst.setActionCommand("MENJAJ");
      menjajtekst.addActionListener(this);
```

```
        setVisible(true);
    }
    void dodajPanele() {
        gornjipanel = new Panel();
        gornjipanel.setBackground(Color.green);
        gornjipanel.add(kraj);
        gornjipanel.add(menjajtekst);
        add("North",gornjipanel);
        donjipanel.setBackground(new Color(0,0,255));
        labela.setForeground(Color.yellow);
        donjipanel.add(labela);
        add("South",donjipanel);
    }
    public void actionPerformed(ActionEvent e) {
      if( e.getActionCommand().equals("KRAJ")) System.exit(0);
      else
        if( e.getActionCommand().equals("MENJAJ")){
          Color nijansa =
                new Color(0,0,(int)(Math.random()*256));
          donjipanel.setBackground(nijansa);
          labela.setBackground(nijansa);
        }
    }
    public static void main(String args[]){
      PrimerPanela prozor1 = new PrimerPanela("Prvi prozor);
      PrimerPanela prozor2 = new PrimerPanela("Drugi prozor);
    }
}
```

Slika 5.18. panel koji sadrži dva panela

Slika 5.19. Izgled dve instance panela koji sadrži dva panela

U primeru datom na slici 5.18. aplikacija otvara dva prozora (prozor1 i prozor2). U frejm se postavljaju dva panela, jedan u severni deo kanvasa metodom `add("North",gornjipanel)`) i drugi u južni deo kanvasa metodom (metod `add("North",donjipanel)`).

Gornji panel zelene pozadinske boje sadrži dva programska dugmeta. Klik na prvo dugme zatvara aplikaciju (u datom primeru zatvaraju se oba otvorena prozora), a klik na drugo dugme odabira slučajnu nijansu plave pozadine donjeg panela i labele.

Donji panel plave pozadinske boje sadrži labelu. Slučajni odabir je rešen metodom `random` klase `Math` (slučajno odabiranje broja od 0 do isključno 1) i kastom na int vrednost `(int)(Math.random()*256).`

Postavljene akcione komande za dugmad su KRAJ i MENJAJ respektivno.

Pošto aplikacija otvara dva prozora, klikom u jednom prozoru na ikonu za zatvaranje tog prozora samo će se osloboditi resursi tog prozora metodom `dispose`(), a drugi prozor ostaje otvoren. Izgled aplikacije `PrimerPanela` dat je na slici 5.19.

Postavljanje boje realizuju metode:

- `setBackground(Color c)` za postavljanje boje pozadine
- `setForeground(Color c)` za postavljanje boje crtanja

Klasa java.awt.Color sadrži konstante predefinisanih vrednosti boja: `Color.black`, `Color.red`, `Color.green`, `Color.blue`, `Color.cyan`, `Color.gray`, `Color.darkGray`, `Color.lightGray`, `Color.magenta`, `Color.orange`, `Color.pink`, `Color.yellow`, `Color.white`.

Boja se definiše kao intenzitet boja: crvene, zelene i plave u celobrojnom opsegu (0-255) ili u normalizovanom opsegu [0,1]. Ovim crna boja ima vrednosti (0,0,0), a bela (255,255,255) u celobrojnom ili (1.0,1.0,1.0) u normalizovanom opsegu.

5.1.11. Raspoređivači komponenti

Za raspoređivanje komponenti u kontejneru koristi se upravljač rasporeda (*layout manager*). Klasa upravljača rasporeda implementira interfejs LayoutManager.

Klase upravljača rasporeda su:

- `FlowLayout` koja komponente u kontejneru raspoređuje po redovim u nizovima sleva-udesno
- `BorderLayout` koja komponente u kontejneru raspoređuje po ivicama i u sredini kontejnera

- `CardLayout` koja komponente u kontejneru raspoređuje kao jednu iza druge (kao špil karata)
- `GridLayout` koja komponente u kontejneru raspoređuje matrično
- `GridBagLayout` koja komponente u kontejneru raspoređuje prema skupu objekata GridBagConstraints
- `null-layout` koja omogućuje da programer eksplicitno odredi poziciju i veličinu komponente

Na slici 5.20. dat je primer `null-layouta` u kome se kreiraju tri programska dugmeta kojima se određuje pozicija i veličina na kanvasu. Pozicija i veličina su određene metodom reshape koja prihvata 4 argumenta (koordinate gornjeg levog ćoška programskog dugmeta te širina i visina programskog dugmeta). Na slivi 5.21. dat je izgled programskih dugmadi raspoređenih `null-layout` raspoređivačem.

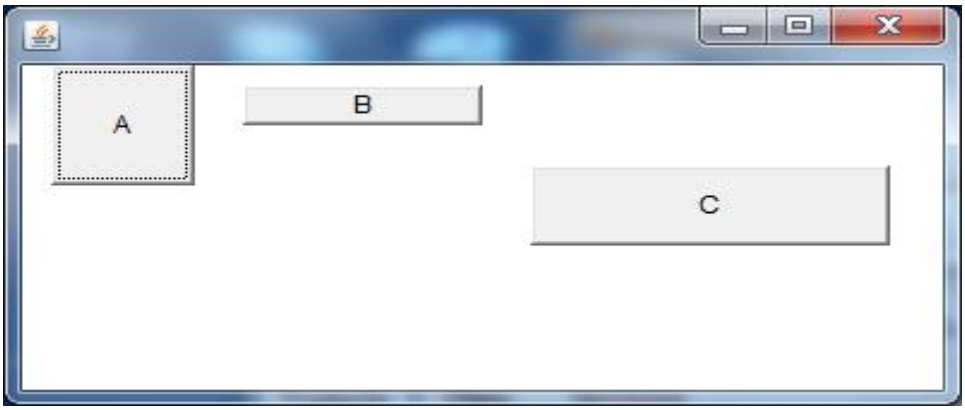
```
import java.awt.*;
import java.awt.event.*;

public class NullLayoutPrimer extends Frame{

        private Button b1, b2, b3;

        NullLayoutPrimer(){
            addWindowListener(new WindowAdapter() {
                  public void windowClosing(WindowEvent e){
                       System.exit(0);
                  }
            });
            setSize(400,200);
            setLayout(null);
            b1 = new Button("A");
            b2 = new Button("B");
            b3 = new Button("C");
            add(b1);
            add(b2);
            add(b3);
            b1.reshape(20, 30, 60, 60);
            b2.reshape(100, 40, 100, 20);
            b3.reshape(220, 80, 150, 40);
            setVisible(true);
        }
        public static void main(String str[]) {
            NullLayoutPrimer nlp = new NullLayoutPrimer();
        }
}
```

Slika 5.20. Raspoređivač null-layout



Slika 5.21. Izgled programskih dugmadi raspoređenih null-layout raspoređivačem

```
import java.awt.*;
import java.awt.event.*;

public class PrimerRasporeda extends Frame {

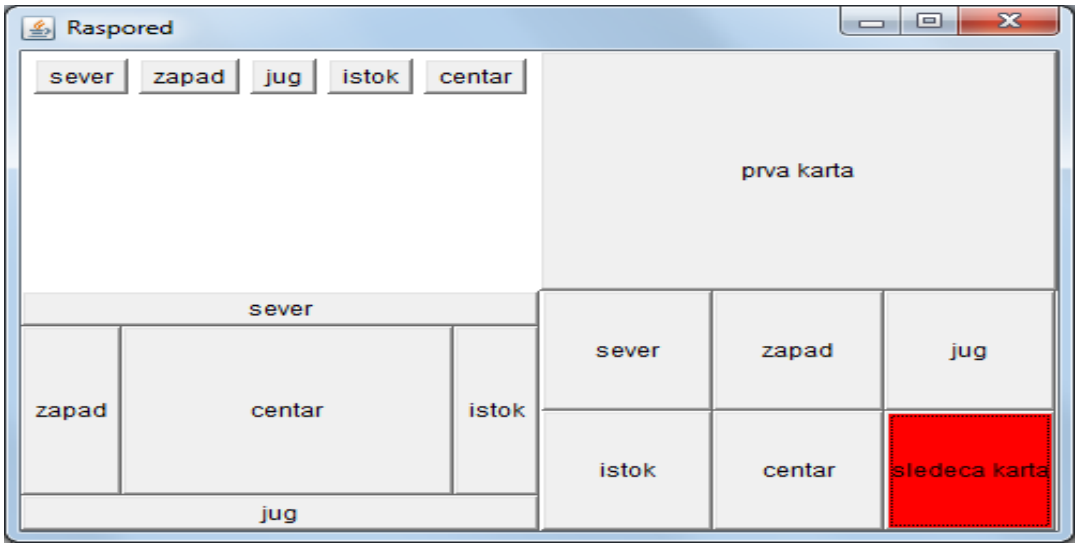
  Button menjajkarte = new Button("sledeca karta");
  CardLayout cardlayout = new CardLayout();
  int indekskarte = 0;
  String nazivkarte[] = {"prva karta","druga karta",
           "treca karta","cetvrta karta","peta karta"};
  Panel flow   = new Panel();
  Panel card   = new Panel();
  Panel border = new Panel();
  Panel grid   = new Panel();

  public PrimerRasporeda() {
    super("Raspored");
    dodajPanele();
    setSize(500,350);
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e){
                      System.exit(0);
      }
    });
    setVisible(true);
  }

  void dodajPanele() {
      setLayout(new GridLayout(2,2));
      dodajTastere(flow);
      card.setLayout(cardlayout);
      for (int i = 0; i < nazivkarte.length; i++)
          card.add(nazivkarte[i],new Button(nazivkarte[i]));
      cardlayout.show(card, nazivkarte[0]);
```

```
      flow.setLayout(new FlowLayout());
      border.setLayout(new BorderLayout());
      grid.setLayout(new GridLayout(2,3));
      dodajTastere(border);
      dodajTastere(grid);
      menjajkarte.addActionListener(new ActionListener(){
         public void actionPerformed(ActionEvent ae){
             if (++indekskarte == nazivkarte.length)
                indekskarte = 0;
             cardlayout.show(card, nazivkarte[indekskarte]);
         }
      });
      menjajkarte.setBackground(Color.red);
      grid.add(menjajkarte);
      add(flow);
      add(card);
      add(border);
      add(grid);
  }
  void dodajTastere(Panel panel){
            panel.add("North" ,new Button("sever" ));
            panel.add("West"  ,new Button("zapad" ));
            panel.add("South" ,new Button("jug"   ));
            panel.add("East"  ,new Button("istok" ));
            panel.add("Center",new Button("centar"));
  }
  public static void main(String args[]){
            PrimerRasporeda prozor = new PrimerRasporeda();
  }
}
```

Slika 5.22. Raspoređivanje komponenti različitim raspoređivačima



Slika 5.23. Izgled aplikacije PrimerRasporeda

U primeru datom na slici 5.22. na kanvasu koji koristi raspoređivač `GridLayout` nalaze se četiri panela raspoređena u matricu 2x2 pri čemu se u prva tri panela nalazi 5 ekranskih tastera, a u četvrtom 6 ekranskih tastera. Izgled je dat na slici 5.23.

Svaki od ova četiri panela ima svoj raspoređivač komponenti po kome su ekranski tasteri raspoređeni. To su `FlowLayout, CardLayout, BorderLayout i GridLayout` respektivno. Panel nosi prefiks naziva svog raspoređivača.

Zatvaranje aplikacije je realizovano implementacijom metode `windowClosing` klase `WindowAdapter.`

Menjanje indeksa programskih dugmadi u drugom panelu realizovan je akcijom na programsko dugme crvene pozadinske boje koje se nalazi u četvrtom panelu. Akcija na događaj klika na ovo dugme je cirkularno menjanje indeksa u metodi `actionPerformed.`

U primeru datom na slici 5.24. pokazuje se kako se koristi `GridBagLayouts` raspoređivač. Raspoređuje se 5 programskih dugmadi tako da prva tri budu u nultom redu (`gridy`) redom u kolonama (`gridx`) 0, 1 i 2 respektivno. Četvrto programsko dugme se nalazi u prvom redu i proteže se na sve tri kolone. Peto programsko dugme je postavljeno dole desno i proteže se na kolone 1 i 2. Postavljanjem atributa za horizontalno popunjavanje na maksimalnu širinu sva programska dugmad će menjanjem veličine frejma menjati i svoju dužinu horizontalno popunjavajući pripadne kolone. Na slikama 5.25, 5.26, 5.27 i 5.28 dati su rasoredi komponenti sa: horizontalnim popunjavanjem i distribucijom prostora između kolona 0.5, , respektivno.

```
import java.awt.*;
```

```
import java.awt.event.*;

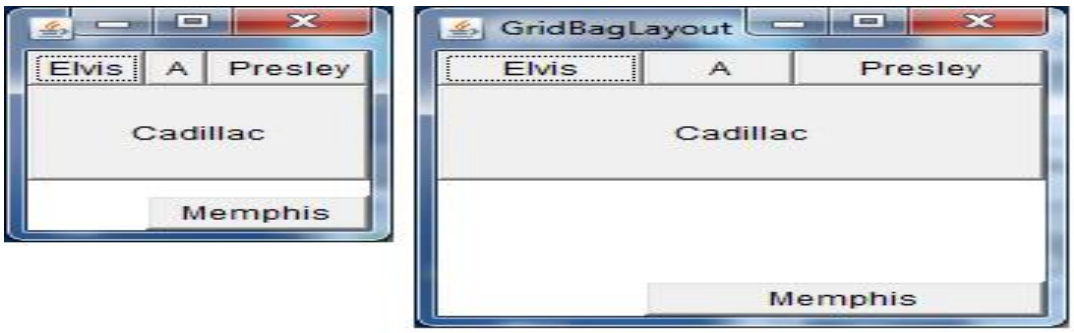
public class GridBagLayoutPrimer extends Frame {

      final boolean shouldFill = true;
      final boolean shouldWeightX = true;

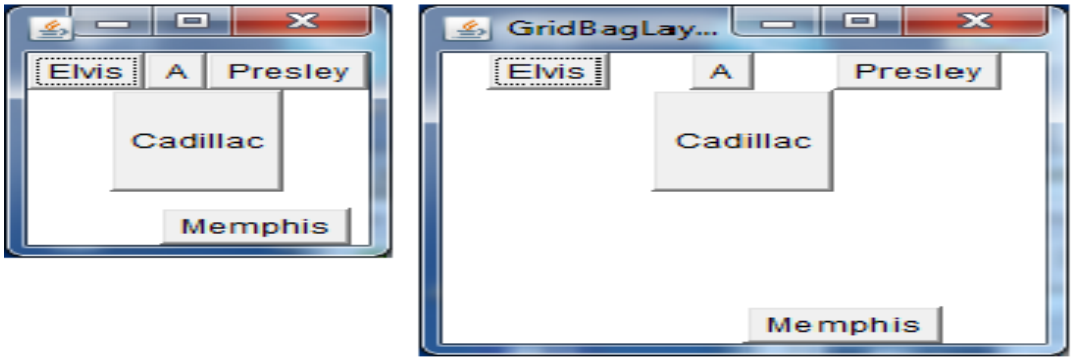
      public GridBagLayoutPrimer() {
          addWindowListener(new WindowAdapter() {
              public void windowClosing(WindowEvent e) {
                     System.exit(0);
            }
        });
        Button button;
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();
        setLayout(gridbag);
        if (shouldFill) {
            //horizontalno popunjavanje na maksimalnu sirinu
            c.fill = GridBagConstraints.HORIZONTAL;
        }
        button = new Button("Elvis");
        if (shouldWeightX) {
            //distribucija prostora izmedju kolona
            c.weightx = 0.5;
        }
        c.gridx = 0;
        c.gridy = 0;
        gridbag.setConstraints(button, c);
        add(button);
        button = new Button("A");
        c.gridx = 1;
        c.gridy = 0;
        gridbag.setConstraints(button, c);
        add(button);
        button = new Button("Presley");
        c.gridx = 2;
        c.gridy = 0;
        gridbag.setConstraints(button, c);
        add(button);
        button = new Button("Cadillac");
        c.ipady = 40;      //povecanje visine
        c.weightx = 0.0;
        c.gridwidth = 3;
        c.gridx = 0;
        c.gridy = 1;
        gridbag.setConstraints(button, c);
```

```
        add(button);
        button = new Button("Memphis");
        c.ipady = 0;       //podrazumevana visina
        c.weighty = 1.0;   //dodavanje vertikalnog razmaka
        c.anchor = GridBagConstraints.SOUTH; //na juznu oblast
        c.insets = new Insets(10,0,0,0);  //razmak oko
                                          //komponente
        c.gridx = 1;       //poravnanje sa dugmetom A
        c.gridwidth = 2;   //sirina 2 kolone
        c.gridy = 2;       //treci red
        gridbag.setConstraints(button, c);
        add(button);
    }
    public static void main(String args[]) {
        GridBagLayoutPrimer win = new GridBagLayoutPrimer();
        win.setTitle("GridBagLayout");
        win.pack();
        win.setVisible(true);
    }
}
```

Slika 5.24. Izgled aplikacije PrimerRasporeda



Slika 5.25. Horizontalno popunjavanje sa distribucijom prostora između kolona 0.5



Slika 5.26. Bez horizontalnog popunjavanja sa distribucijom prostora između kolona 0.5

Slika 5.27. Sa horizontalnim popunjavanjem i podrazumevanom distribucijom prostora između kolona

Slika 5.28. Bez horizontalnog popunjavanja i podrazumevanom distribucijom prostora između kolona

5.1.12. Polje za potvrdu i radio-dugmad

Pomenuto je da klasa `Checkbox` služi za kreiranje polja za potvrdu i radio-dugmadi.

Polje za potvrdu ima atribute za naziv i stanje koji se mogu modifikovati metodama klase `Checkbox`. Promena stanja polja za potvrdu izazvaće događaj `ItemEvent` koji se obrađuje metodom `itemStateChanged`(ItemEvent).

Radio-dugmad se kreiraju tako što se pozove konstruktor klase `Checkbox` kome se prosleđuju: naziv radio-dugmeta, referenca na zajednički objekat klase `CheckboxGroup` i inicjalno stanje koje pokazuje da li je radio-dugme uključeno ili ne. Ovim su radio-dugmad grupisana u `CheckboxGroup-u` (slika 5.29).

```
import java.awt.*;
import java.awt.event.*;

public class PrimerCheckbox
```

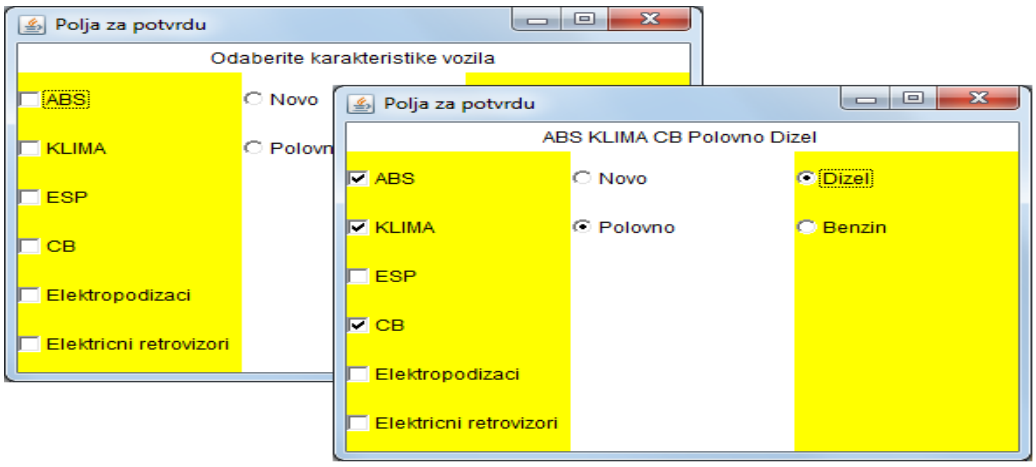
```
            extends Frame
            implements ItemListener {
Label labela = new Label("Odaberite karakteristike vozila");
CheckboxGroup grupanovo   = new CheckboxGroup();
CheckboxGroup grupagorivo = new CheckboxGroup();
Checkbox potvrde[] = {
    new Checkbox("ABS"),
    new Checkbox("KLIMA"),
    new Checkbox("ESP"),
    new Checkbox("CB"),
    new Checkbox("Elektropodizaci"),
    new Checkbox("Elektricni retrovizori"),
    new Checkbox("Novo"   ,grupanovo ,false),
    new Checkbox("Polovno",grupanovo ,false),
    new Checkbox("Dizel" ,grupagorivo,false),
    new Checkbox("Benzin",grupagorivo,false)
};
public PrimerCheckbox() {
  super("Polja za potvrdu");
  init();
  setSize(400,300);
  addWindowListener(new WindowAdapter(){
      public void windowClosing(WindowEvent we){
          System.exit(0);
      }
  });
  setVisible(true);
}
void init() {
  add("North",labela);
  labela.setAlignment(Label.CENTER);
  Panel panel = new Panel();
  panel.setLayout(new GridLayout(1,3));
  Panel panelpotvrda = new Panel();
  panelpotvrda.setLayout(new GridLayout(6,1));
  panelpotvrda.setBackground(Color.yellow);
  Panel panelradiodugmadi_1 = new Panel();
  panelradiodugmadi_1.setLayout(new GridLayout(6,1));
  Panel panelradiodugmadi_2 = new Panel();
  panelradiodugmadi_2.setLayout(new GridLayout(6,1));
  panelradiodugmadi_2.setBackground(Color.yellow);
  for(int i=0;i<potvrde.length;++i)
      potvrde[i].addItemListener(this);
  for(int i=0;i<potvrde.length-4;++i)
      panelpotvrda.add(potvrde[i]);
```

```
    for(int i=potvrde.length-4;i<potvrde.length-2;++i)
        panelradiodugmadi_1.add(potvrde[i]);
    for(int i=potvrde.length-2;i<potvrde.length;++i)
        panelradiodugmadi_2.add(potvrde[i]);
    panel.add(panelpotvrda);
    panel.add(panelradiodugmadi_1);
    panel.add(panelradiodugmadi_2);
    add("Center",panel);
  }
  public void itemStateChanged(ItemEvent e) {
      String tekst = "";
      for(int i=0;i<potvrde.length;++i) {
            if(potvrde[i].getState())
            tekst+=potvrde[i].getLabel()+" ";
      }
      labela.setText(tekst);
  }
  public static void main(String args[]){
      PrimerCheckbox prozor = new PrimerCheckbox ();
  }
}
```

Slika 5.29. Polja za potvrdu i radio dugmad

U primeru datom na slici 5.29. kreirana su tri panela od kojih jedan sadrži polje dugmadi za potvrdu, a preostala dva po dva radio-dugmeta. Izabrane vrednosti se prikazuju u tekstu labele smeštene na sever kanvasa.

Svi elementi niza potvrde se osluškuju u metodi `itemStateChanged` koja je implementirana u klasi `PrimerCheckbox`. Kada nastane događaj provere se stanja svih elemenata niza i prikažu nazivi odabranih elemenata (slika 5.30). Provera stanja elemenata Checkbox vrši se metodom getState() koja vraća logičku vrednost da li je dati element čekiran.



Slika 5.30. Izgled aplikacije PrimerCheckbox

5.1.13. Klase List i Choice

Klasa `List` omogućuje realizaciju liste iz koje se odabira jedan ili više redova. Metodi klase `List` mogu da daju status elemenata te da ih modifikuju.

Za realizaciju padajuće liste koristi se klasa `Choice`. Kod padajuće liste moguće je odabrati samo jedan red. Metodi klase `List` mogu da daju status elemenata te da ih modifikuju.

```
import java.awt.*;
import java.awt.event.*;

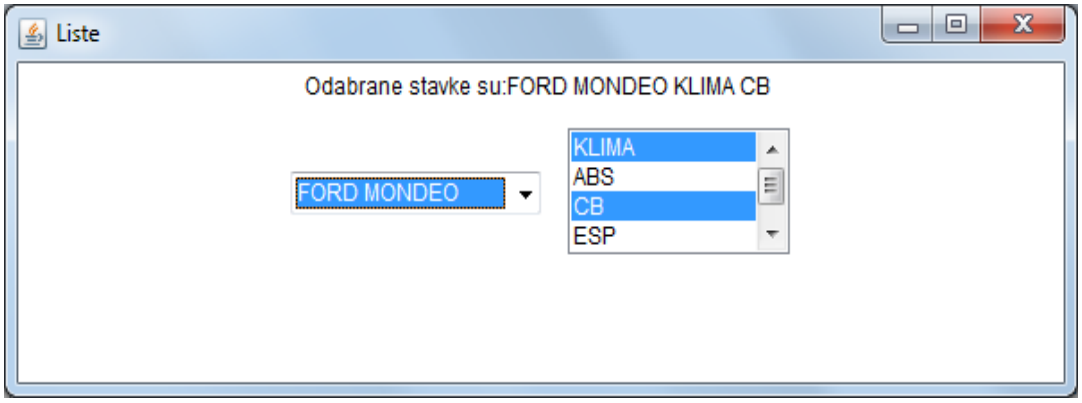
public class PrimerListChoice
             extends Frame
             implements ItemListener{
      Label labela = new Label("Pocetni tekst");
      Choice izbor = new Choice();
      List lista = new List(4,true);

      public PrimerListChoice() {
            super("Liste");
            dodajKomponente();
            setSize(600,200);
            setVisible(true);
            itemStateChanged(null);
      }

      void dodajKomponente() {
            add("North",labela);
            labela.setAlignment(Label.CENTER);
            izbor.addItem("FORD MUSTANG");
            izbor.addItem("FORD MONDEO");
            izbor.addItem("FORD FOCUS");
            izbor.addItem("Zatvaranje aplikacije");
            izbor.addItemListener(this);
            lista.add("KLIMA");
            lista.add("ABS");
            lista.add("CB");
            lista.add("ESP");
            lista.add("Navigacija");
            lista.addItemListener(this);
            Panel panel  = new Panel(),
                  panel1 = new Panel(),
                  panel2 = new Panel();
            panel1.add(izbor);
```

```
            panel2.add(lista);
            panel.add(panel1);
            panel.add(panel2);
            add("Center",panel);
      }
      public void itemStateChanged(ItemEvent e) {
        String tekst = "Odabrane stavke su:";
        tekst+=izbor.getSelectedItem() + " ";
        if(izbor.getSelectedItem().equals
                                    ("Zatvaranje aplikacije"))
           System.exit(0);
        for(int i=0;i<3;++i)
           if(lista.isIndexSelected(i))
              tekst += lista.getItem(i) + " ";
        labela.setText(tekst);
      }
      public static void main(String args[]){
        PrimerListChoice prozor = new PrimerListChoice();
      }
}
```

Slika 5.31. Korišćenje liste i padajuće liste



Slika 5.32. Izgled aplikacije PrimerListChoice

U primeru na slci 5.31. je data aplikacija koja u labeli smeštenoj na "sever" frejma ispisuje odabranu stavku u padajućoj listi i odabrane stavke u listi.

Događaji kontrola `List` i `Choice` se obrađuju u metodi osluškivača `itemStateChanged`. Dodavanje stavki u padajuću listu i listu omogućuju metode `addItem` i `add` respektivno. Dohvatanje odabrane stavke padajuće liste omogućuje metoda `getSelectedItem()` dok je za dohvatanje selektovanih stavki liste potrebno ispitati za svaku stavku da li je njen indeks selektovan metodom `isIndexSelected(index)`, a ako jeste, onda stavku dohvatiti metodom `getItem(index)`.

Kada se u padajućoj listi odabere stavka "Zatvaranje aplikacije" aplikacija završava rad. Izgled aplikacije dat je na slici 5.32.

5.1.14. Polje za tekst i oblast za tekst

Iz klase `TextComponent` izvedene su klase `TextField` i TextArea.

Klasom `TextField` realizuje se prikaz i unos jedne linije teksta pri čemu je metodom `setEchoCharacter()` moguće definisati znak koji će se prikazivati pri kucanju lozinke.

Klasa `TextArea` realizuje prikaz i unos više linija teksta pri čemu je metodom `setEditable(false)` tekst samo za čitanje. Prikaz oblasti teksta može biti i sa horizontalnim i vertikalanim klizačem za navigaciju po oblasti teksta.

```
import java.awt.*;
import java.awt.event.*;

public class PrimerTekst
             extends Frame
             implements ActionListener{
      int suma;
      TextField tekstpolje =
                new TextField("Unesite integer ili tekst.");
      TextArea tekstoblast =
                new TextArea("Prikazuje se suma unesenih brojeva
                ili tekst");

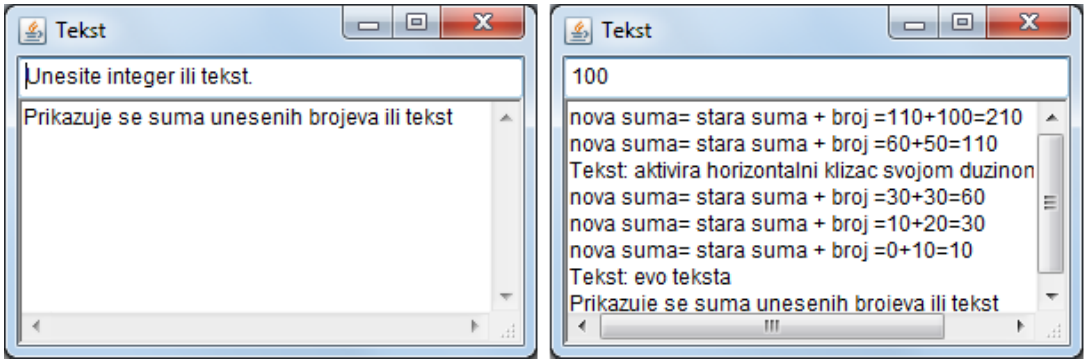
      public PrimerTekst() {
            super("Tekst");
              suma = 0;
            addWindowListener(new WindowAdapter(){
              public void windowClosing(WindowEvent we){
                System.exit(0);
             }
            });
            dodajKomponente();
            setSize(300,200);
            setVisible(true);
      }

      void dodajKomponente() {
            add("North",tekstpolje);
            add("Center",tekstoblast);
            tekstpolje.addActionListener(this);
      }

      public void actionPerformed(ActionEvent e) {
        String tekst = tekstpolje.getText();
```

```
        if(tekst.equals("Oslobodi resurse")) dispose();
        try{
          int broj = Integer.parseInt(tekst);
          tekstoblast.insert("nova suma= stara suma + broj
               ="+suma+"+"+broj+"="+(suma+=broj)+"\n",0);
        }
        catch(Exception ex){
                 tekstoblast.insert("Tekst: "+tekst+"\n",0);
        }
      }
      public static void main(String args[]){
            PrimerTekst prozor1 = new PrimerTekst ();
            PrimerTekst prozor2 = new PrimerTekst ();
      }
}
```

Slika 5.33. Korišćenje klasa TextField i TextArea



Slika 5.34. Izgled aplikacije PrimerTekst

U primeru datom na slici 5.33. u polje za tekst se unosi integer ili tekst. Aplikacija otvara dva prozora (`prozor1` i `prozor2`) tako da će ubacivanjem teksta "`Zatvori resurse`" dovesti do zatvaranja resursa i nestajanje pripadnog prozora sa ekrana.

Ako je bio unesen integer, onda se u vrh oblasti za tekst (indeks 0) ubacuje informacija o sumi do tada unesenih brojeva. U slučaju da se unese tekst on će biti samo insertovan na vrh oblasti za tekst. Videti sliku 5.34.

Nakon potvrde unosa teksta u polje za tekst (enter) aktivira se događaj `ActionEvent` koji se obrađuje u metodi `actionPerformed` klase koja implementira interfejs `ActionListener` i koja je registrovana da obrađuje događaj koji je izazvala ova komponenta.

Zatvaranje aplikacije realizovano je klikom na ikonu za zatvaranje prozora.

5.2. Biblioteka Swing

Biblioteka kompeneti AWT bazirana je na korišćenju dostupnih komponenti GUI-a na datom operativnom sistemu. Tako klase AWT-a za Windows koriste awt.dll dinamičku biblioteku. Pošto je Java platformski prenosiva odabran je presečni skup komponenti GUI-a različitih operativnih sistema što je dovelo do siromašnog skupa GUI komponenti.

U prevazilaženju ovog problema kreiran je novi paket GUI komponenti javax.swing. Prednost je bila što je swing napisan u Javi čime je ostala prenosivost koda uz zadržavanje definisanog izgleda na svim platformama uz "neograničenost" broja komponenti koje će biti u biblioteci.

Komponente Swing biblioteke se mogu smestiti samo unutar kontejnera. Komponente Swing korisničkog interfejsa po pravilu počinju velikim slovom J.

U prilozima, u tabeli A4. date su najčešće korišćene Swing komponente.

5.2.1. Definisanje izgleda Swing aplikacije

Definisanje izgleda swing aplikacije odnosi se na odabranu temu prikaza GUI komponenti. Ove teme poznate kao *look-and-feel* moduli izgledaju shodno svome nazivu. Tri često korišćene teme su:

- Windows
  - komponente izgledaju kao Windows komponente
- Motif
  - komponente izgledaju kao GUI okruženje na Linux-u
- Metal
  - Java izgled komponenti

Promena teme može se promeniti i za vreme izvršavanja programa. Od verzije 1.2 Swing biblioteka je proglašena standardnom za razvoj korisničkog interfejsa u Java aplikacijama, a AWT je zadržan zbog kompatibilnosti. Swing je deo biblioteke JFC (Java Foundation Classes).

Sledi primer menjanja look-and-feel teme:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class PrimerLookAndFeel
             extends JFrame
             implements ActionListener {
  private int tipsminke;
  private String str;
```

```
  private JButton jdugme;

  public PrimerLookAndFeel() {
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      tipsminke = 0;
      str = "";
      jdugme = new JButton("Promeni izgled");
      getContentPane().setLayout(new GridLayout(2,1));
      getContentPane().add("North",jdugme);
      //ili getContentPane().add(jdugme,BorderLayout.NORTH);
      jdugme.addActionListener(this);
      setSize(400,100);
      setTitle("Prikaz look-and-feel");
      setVisible(true);
  }

  public void actionPerformed(ActionEvent e) {
    tipsminke++; tipsminke%=3;
    switch(tipsminke)
    {
      case 0: str =
            "javax.swing.plaf.metal.MetalLookAndFeel";break;
      case 1: str =
 "com.sun.java.swing.plaf.windows.WindowsLookAndFeel";break;
      default:str =
           "com.sun.java.swing.plaf.motif.MotifLookAndFeel";
    }
    jdugme.setText(" ovo je "+str);
    try {
            UIManager.setLookAndFeel(str);
            SwingUtilities.updateComponentTreeUI(this);
      } catch (Exception excep) {}
  }

  public static void main(String[] args) {
      PrimerLookAndFeel plaf = new PrimerLookAndFeel();
  }
}
```

Slika 5.35. Postavljanje LookAndFeel teme

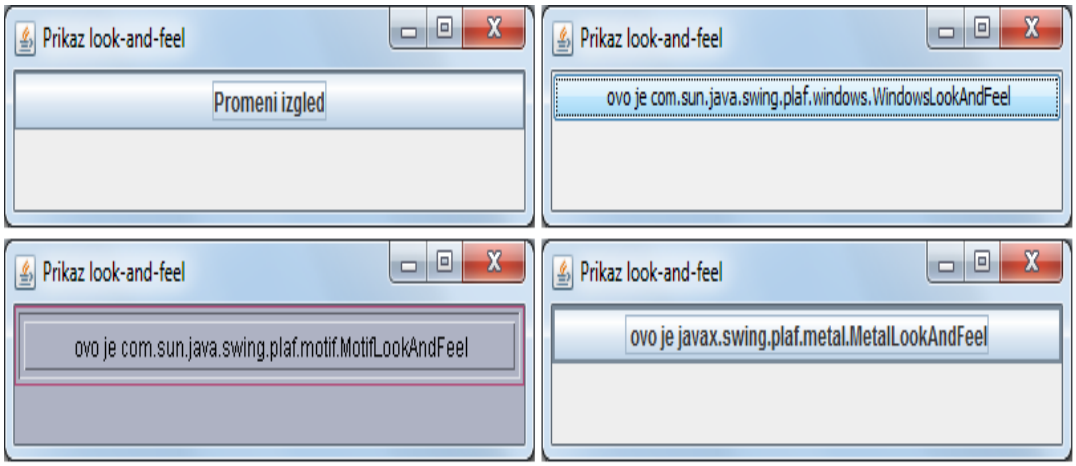
U primeru datom na slici 5.35. kreirana je klasa `PrimerLookAndFeel` izvedena iz klase `JFrame` koja pripada paketu `javax.swing` (obratiti pažnju na prefiks `javax`). Naziv ovog frejma postavlja se prosleđivanjem naziva metodi `setTitle(String)`.

Atributi klase su tipa `int`, `String` i `JButton`. Celi broj se koristi kao selektor teme koja se primenjuje na celu aplikaciju. Atribut tipa `String` koriti se ze prosleđivanje parametra metodi `setLookAndFeel` klase `UIManager`. Ovaj

parametar određuje koja će tema biti upotrebljena za prikaz aplikacije. Programsko dugme klase `JButton` izaziva događaje koje obrađuje metoda `actionPerformed` kao i ranije. Klikom miša na ovo programsko dugme menjaće se cirkularno tri teme (`windows`, `motif`, `metal`). Videti sliku 5.36.

Ažuriranje promene teme relizovano je metodom `updateComponentTreeUI` klase `SwingUtilities`.

Postavljanje programskog dugmeta u kontejner prozora obavlja se metodom `add` kontejnera prozora. Kontejner prozora se dohvata metodom `getContentPane()` klase `JFrame`.



Slika 5.36. Menjanje teme LookAndFeel

Zatvaranje aplikacije je relizovano u samom `JFrame`-u koji reaguje na klik na ikonu za zatvaranje aplikacije na način koji odgovara stvarnom argumentu metode:

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Raspoređivači komponenti se ponašaju isto kao i ranije pri čemu je podrazumevani raspoređivač `BorderLayout`. Odgovarajuće `Layout` klase se i dalje nalaze u paketu `java.awt`.

5.2.2. Događaji i osluškivači

Događaji i osluškivači se ponašaju isto kao i u AWT-u jer su i dalje u paketima `java.awt` i `java.awt.event`. Pridruživanje osluškivača komponenti koja može izazvati adekvatan događaj u Swing GUI-u je kao i u AWT-u.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PrimerJFrame extends JFrame {
```

```
  private JButton povecanje =
          new JButton("Povecanje JFrame-a");
  private JButton umanjenje =
          new JButton("Smanjenje JFrame-a");
  private int delta = 20;
  public PrimerJFrame() {
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(580+delta, 130+delta);
    setTitle("Povecanje i smanjenje JFrame-a");
    getContentPane().setLayout(new FlowLayout());
    getContentPane().add(povecanje);
    getContentPane().add(umanjenje);
    povecanje.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ev) {
        delta+=20;
        setSize(580+delta,130+delta);
      }
    });
    umanjenje.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ev) {
        if(delta>20)delta-=20;
        setSize(580+delta,130+delta);
      }
    });
  }
  public static void main(String argumenti[]){
      PrimerJFrame pjf = new PrimerJFrame();
      pjf.setVisible(true);
  }
 }
}
```

Slika 5.37. Menjanje veličine JFramea

Slika 5.38. Menjanje veličine JFramea

U primeru datom na slici 5.37. frejm (JFrame) sadrži dva programska dugmeta

povecanje i smanjenje. Preko neimenovane klase implementirano je da klik miša na programsko dugme povecanje povećava trenutnu veličinu frejma, dok klik miša na programsko dugme smanjenje veličinu frejma do minimalne veličine frejma koja odgovara početnoj veličini frejma. Komponente su postavljene u kontejner prozora koji je dohvaćen metodom getContentPane. Izgled aplikacije dati je na slici 5.38.

5.2.3. Klasa KeyAdapter

U primeru datom na slici 5.39. aplikacija u matričnom raspoređivaču postavlja dve komponente: labelu i tekst polje. Kucanjem reči u tekstu labele se ispisuje da se reč upravo kuca dok će kucanje razmaka učiniti da se celo tekst polje obriše što će se evidentirati u tekstu labele.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PrimerJTextField extends JFrame {

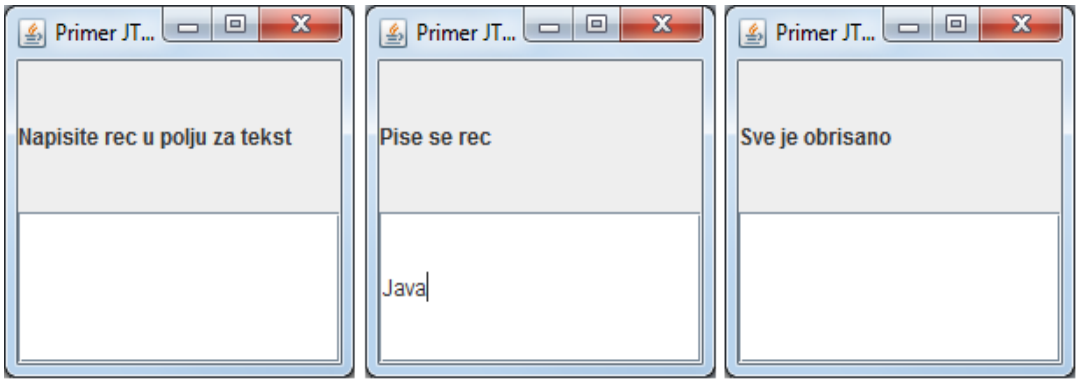
 JTextField tf = new JTextField(30);
 JLabel l = new JLabel("Napisite rec u polju za tekst");

 public PrimerJTextField() {
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(200, 200);
    setTitle("Primer JTextField");
    getContentPane().setLayout(new GridLayout(2,1));
    getContentPane().add(l);
    getContentPane().add(tf);
    tf.addKeyListener(new Obradjivac());
  }

  class Obradjivac extends KeyAdapter {
    public void keyReleased(KeyEvent e) {
        if(e.getKeyCode() == KeyEvent.VK_SPACE){
              l.setText("Sve je obrisano");
              tf.setText(null);
        }
        else  l.setText("Pise se rec");
    }
  }

  public static void main(String []arg){
     PrimerJTextField pjtf = new PrimerJTextField();
     pjtf.setVisible(true);
  }
}
```

Slika 5.39. Korišćenje klase KeyAdapter



Slika 5.40. Izgled aplikacije PrimerJTextField

Osluškivač je realizovan pomoću unutrašnje klase `Obradjivac` koja nasleđuje klasu `KeyAdapter`. Klasa `KeyAdapter` implementira na prazno metode interfejsa `KeyListener`. Redefinisana je metoda `keyReleased` koja reaguje na otpuštanje tastera. Tekst polju `tf` je pridružen oslučkivač tastature. Ispituje se da li je otpušteni taster bio razmak (`KeyEvent.VK_SPACE`) te ako je bio, onda će se u labeli ispisati "`Sve je obrisano`", inače će u labeli pisati "`Pise se rec`". Na slici 5.40. dat je izgled ove aplikacije.

5.2.4. Interfejs CaretListener

U primeru na slici 5.41. pokazana je upotreba oslučkivača kursora. Aplikacija `PrimerOsluskivacaKursora` sadrži labelu u koju se upisuje informacija o poziciji tekstualnog kursora u oblasti za tekst.

```
import java.awt.*;
import javax.swing.event.*;
import javax.swing.*;

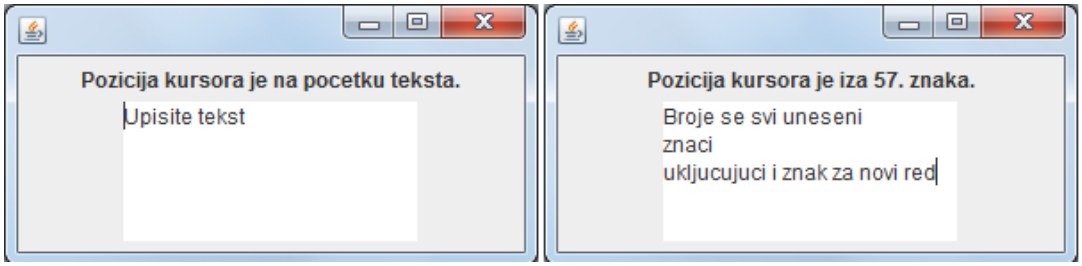
public class PrimerOsluskivacaKursora extends JFrame {
  JLabel jlabel;
  JTextArea jtextarea;

  public PrimerOsluskivacaKursora() {
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(300, 150);
    Container cp = getContentPane();
    cp.setLayout(new FlowLayout());
    jlabel = new JLabel("Pozicija kursora je na pocetku teksta.");
    cp.add(jlabel);
    jtextarea = new JTextArea("Upisite tekst", 5, 15);
    jtextarea.addCaretListener(new Osluskivac());
    cp.add(jtextarea);
  }
```

```
  class Osluskivac implements CaretListener {
    public void caretUpdate(CaretEvent e) {
      if(e.getDot()!=0)
        jlabel.setText("Pozicija  kursora  je  iza  "+e.getDot()+".
znaka.");
      else
        jlabel.setText("Pozicija kursora je na pocetku teksta.");
    }
  }

  public static void main(String []arg){
     PrimerOsluskivacaKursora pok = new PrimerOsluskivacaKursora();
     pok.setVisible(true);
  }
}
```

Slika 5.41. Osluškivač kursora



Slika 5.42. Izgled aplikacije koja osluškuje kursor

Pomeranje kursora osluškuje klasa koja implementira `CaretListener` interfejs, a obrada ovog događaja se obavlja u metodi `caretUpdate`. Metoda `getDot` klase `CaretEvent` vraća novi položaj kursora, tako da se u labelu može upisati iza kog karaktera u oblasti za tekst se nalazi kursor (slika 5.42).

5.2.5. Klase JCheckBox i JRadioButton

U `AWT`-u je za implementaciju radio-dugmadi i polja za potvrdu korišćena klasa Checkbox. U `Swing`-u postoje posebne klase: `JCheckBox` i `JRadioButton`.

Za grupisanje radio-dugmadi koristi se klasa `ButtonGroup` (bez prefiksa J) koja metodom `add` dodaje radio-dugmad u datu grupu.

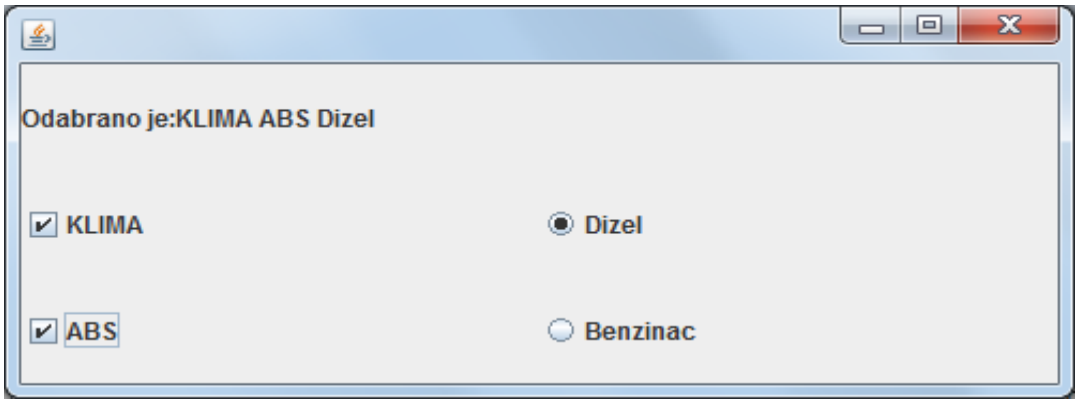
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PrimerJCheckBoxJRadioButton extends JFrame {

  JLabel jlabel = new JLabel("bilo sta");;
  JCheckBox jcheckbox_1 = new JCheckBox("KLIMA");
  JCheckBox jcheckbox_2 = new JCheckBox("ABS");
  ButtonGroup group = new ButtonGroup();
```

```
  JRadioButton radiodugme_1 =
               new JRadioButton("Dizel", true);
  JRadioButton radiodugme_2 =
               new JRadioButton("Benzinac", false);
  public PrimerJCheckBoxJRadioButton() {
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(500, 200);
    Osluskivac osluskivac = new Osluskivac();
    osluskivac.itemStateChanged(null);
    Container cp = getContentPane();
    cp.setLayout(new GridLayout(3,2));
    jcheckbox_1.addItemListener(new Osluskivac());
    jcheckbox_2.addItemListener(new Osluskivac());
    group.add(radiodugme_1); // dodaju se radio
    group.add(radiodugme_2); // button-i u grupu
    radiodugme_1.addItemListener(osluskivac);
    radiodugme_2.addItemListener(osluskivac);
    cp.add(jlabel);
    cp.add(new JPanel()); //za prazno polje GridLayouta
    cp.add(jcheckbox_1);
    cp.add(radiodugme_1);
    cp.add(jcheckbox_2);
    cp.add(radiodugme_2);
  }
  class Osluskivac implements ItemListener {
    public void itemStateChanged(ItemEvent e) {
        String str="Odabrano je:";
        if(jcheckbox_1.isSelected())
            str+=(jcheckbox_1.getText()+" ");
        if(jcheckbox_2.isSelected())
            str+=(jcheckbox_2.getText()+" ");
        if(radiodugme_1.isSelected())
                    str+=(radiodugme_1.getText()+" ");
        if(radiodugme_2.isSelected())
                    str+=(radiodugme_2.getText()+" ");
        jlabel.setText(str);
    }
  }
  public static void main(String []arg){
     PrimerJCheckBoxJRadioButton pok =
              new PrimerJCheckBoxJRadioButton();
     pok.setVisible(true);
  }
}
```

Slika 5.43. Upotreba klasa JCheckBox i JRadioButton



Slika 5.44. Izgled aplikacije PrimerJCheckBoxJRadioButton

U primeru datom na slici 5.43. iskorišćeno je popunjavanje ćelije raspoređivača GridLayout komponentom `JPanel` (samo da popuni ćeliju) kako bi se dobio raspored komponenti prikazan na slici 5.44. U klasi `Osluskivac` implementiran je interfejs `ItemListener` koji ima metodu `itemStateChanged` zaduženu za obradu događaja promene stanja polja za potvrdu i stanja radio-dugmadi. Za ove komponente poziva se metoda isSelected() čime se dobija logički odgovor da li je data komponenta selektovana.

Tekst labele opisuje koja su polja potvrde odabrana i koje je radio-dugme u radio grupi uključeno.

5.2.6. Klasa JComboBox

U `Swingu` se za realizaciju padajuće liste koristi klasa `JComboBox`. I ovde se koristi interfejs `ItemListener` za osluškivanje događaja navedene komponente. U metodi `itemStateChanged` se koristi ispitivanje da li je instanca klase izvora događaja tipa `JComboBox` te da bi se postavio tekst labele koji odgovara odabranom tekstu iz padajuće liste. Odabrana stavka se dohvata metodom `getSelectedItem` (slika 5.45).

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

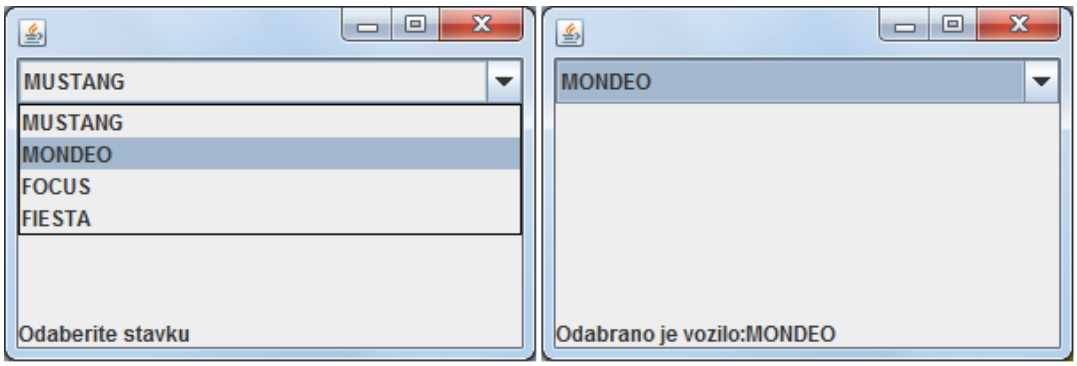
public class PrimerJComboBox extends JFrame {

    String[] items = {"MUSTANG", "MONDEO",
                      "FOCUS", "FIESTA"};
    JComboBox jcombobox = new JComboBox();
    JLabel jlabel = new JLabel("Odaberite stavku");

    public PrimerJComboBox() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300, 200);
```

```
        for (int i = 0; i < items.length; i++) {
            jcombobox.addItem(items[i]);
        }
        jcombobox.addItemListener(new Osluskivac());
        Container cp = getContentPane();
        cp.setLayout(new BorderLayout());
        cp.add(jcombobox,"North");
        cp.add(new JPanel(),"Center");
        cp.add(jlabel,"South");
    }
    class Osluskivac implements ItemListener {
        public void itemStateChanged(ItemEvent e) {
            if (e.getSource() instanceof JComboBox) {
                jlabel.setText("Odabrano je vozilo:"+
                       (String)jcombobox.getSelectedItem());
            }
        }
    }
    public static void main(String []arg){
     PrimerJComboBox pjcb = new PrimerJComboBox();
     pjcb.setVisible(true);
}
```

Slika 5.45. Aplikacija koja korsiti JComboBox



Slika 5.46. Izgled aplikacije PrimerJComboBox

Izgled aplikacije koja koristi klasu JComboBox dat je na slici 5.46. U primeru je inicjalno postavljen tekst na labeli da se odabere stavka (automobil) iz padajuće liste, a onda po odabiranju u labeli se ispisuje koje je vozilo odabrano.

5.2.7. Klasa JOptionPane

Klasa `JoptionPane` sadrži statičke metode kojima realizuje jednostavne dijaloge u `swing`-u.

Najčešće se koristi kao:

- dijalog za unos teksta

```
String str = JOptionPane.showInputDialog(null,
                                "Unesite ime!");
```

- dijalog prikaz poruke

```
str+=" dobrodosli !";
JOptionPane.showMessageDialog(null,str);
```

- dijalog izbora

```
int odgovor = JOptionPane.showConfirmDialog(null,
                    "odustajete li ?",
                    "NAZIV DIJALOGA",
                    JOptionPane.YES_NO_OPTION);
```

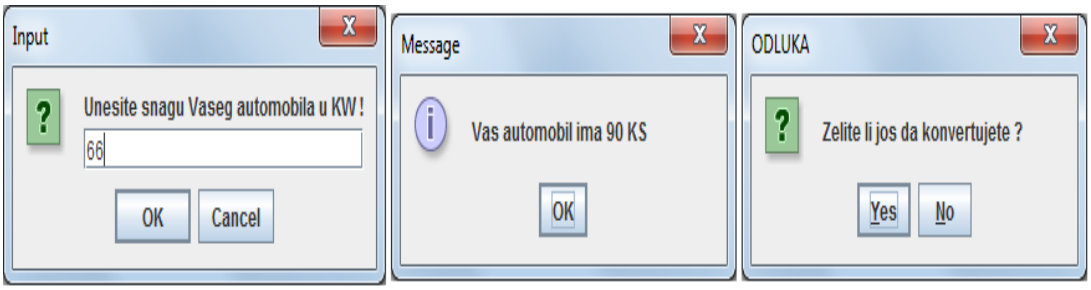
Na slici Sledi primer korišćenja klase JOptionPane.

```
import javax.swing.*;

public class Jop {

  public static void main (String[]args){
      String strsnaga="";
      double snaga=0;
      int odgovor;
      do{
        strsnaga = OptionPane.showInputDialog(null,
                    "Unesite snagu Vaseg automobila u KW !");
       try{
         snaga = (Double.parseDouble(strsnaga))*1.3636;
         JOptionPane.showMessageDialog(null,
              "Vas automobil ima "+Math.round(snaga)+" KS");
       }
       catch(Exception ex){
            System.out.println(ex);
       }
       odgovor = JOptionPane.showConfirmDialog(null,
                        "Zelite li jos da konvertujete ?",
                        "ODLUKA",
                        JOptionPane.YES_NO_OPTION);
      }while(odgovor==0);
  }
}
```

Slika 5.47. Upotreba klase JOptionPane



Slika 5.48. Konverzija KW u KS

U primeru datom na slici 5.47. ilustrovana je upotreba tri vrste dijaloga realizovanih pomoću `JOptionPane` klase. Korišćene statičke metode su: `showInputDialog`, `showMessageDialog` i `showConfirmDialog` za dijaloge unosa teksta, prikaza poruke i izbora respektivno. Prvi dijalog traži da se unese snaga automobila u KW, zatim se prikazuje dijalog koji daje informaciju o broju konjskih snaga koji odgovara unesenoj vrednosti u KW i nakon toga se prikazuje dijalog izbora da li da se opet ide na unos i konverziju ili je to kraj aplikacije (slika 5.48). Programska dugmad na dijalogu su redom indeksirana počevši od nule.

5.2.8. Horizontalni i vertikalni klizači

U programima se često koriste horizontalni i vertikalni klizači. Za implementaciju ovih komponenti koristi se klasa `Scrollbar` i interfejs `AdjustmentListener` (slika 5.49).

```
import java.awt.*;
import java.awt.event.*;

public class Jsb extends Frame {

    Label label;
    // osluskivac kao unutrasnja klasa

    class MyAdjustmentListener
          implements AdjustmentListener {

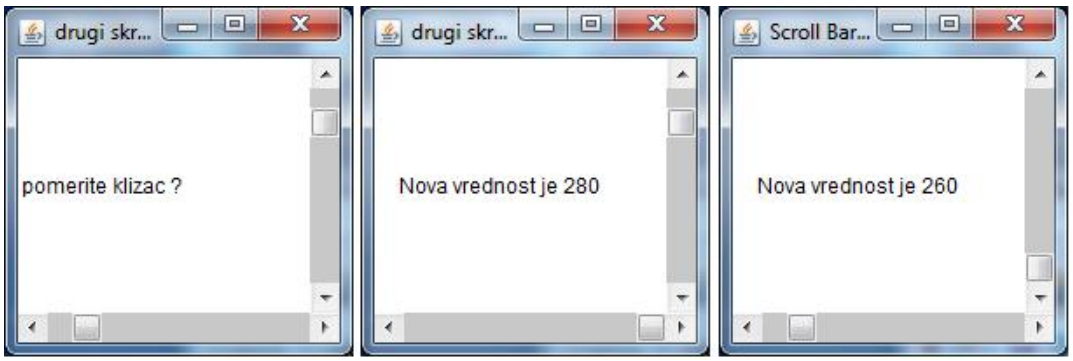
    public void adjustmentValueChanged(AdjustmentEvent e) {
         label.setText("    Nova vrednost je " + e.getValue() +
                       "      ");
         repaint();
        }
    }

    public Jsb(String naslov) {
        super(naslov);
```

```
        setSize(200,200);
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent we){
                  dispose();
            }
        });
        label=new Label("pomerite klizac ?");
        //setLayout(new BorderLayout());
        Scrollbar hbar = new Scrollbar(
                     Scrollbar.HORIZONTAL, 30, 20, 0, 300);
                     //start,korak,min,max
        Scrollbar vbar = new Scrollbar(
                     Scrollbar.VERTICAL, 30, 40, 0, 300);
        hbar.setUnitIncrement(2);
        hbar.setBlockIncrement(1);
        hbar.addAdjustmentListener(
             new MyAdjustmentListener());
        vbar.addAdjustmentListener(
             new MyAdjustmentListener());
        add(hbar, BorderLayout.SOUTH);
        add(vbar, BorderLayout.EAST);
        add(label, BorderLayout.CENTER);
        setVisible(true);
    }
    public static void main(String s[]) {
        Jsb j = new Jsb("Scrollbar");
    }
}
```

Slika 5.49. Horizontalni i vertikalni klizači

Interfejs `AdjustmentListener` ima metodu `adjustmentValueChanged` koja u imlementaciji u datom programu postavlja tekst labele koji odgovara vrednosti pozicije klizača (slika 5.50). Kreiranjem ScrollBara postavljaju se: orijentacija, startna vrednost klizača, korak klizača, minimalna i na kraju maksimalna vrednost opsega koju klizač obuhvata.



Slika 5.50. Horizontalni i vertikalni klizači

5.2.9. Klasa JScrollPane

Sledeći primer dat na slici 5.51. ilustruje korišćenje klase `JScrollPane`. Kanvas je podeljen na matricu koja ima dva reda i jednu kolonu. U prvom redu se prikazuje slika koja se može skrolovati (slika 5.52). Da bi se element dodao u `JScrollPane` koristi se prvo metoda `getViewport()`, a onda se poziva metoda add kojoj se prosleđuje element koji se dodaje u `JScrollPane`. Element koji je u programu dodat je labela koja je inicijalizovana referencom na objekat tipa ImageIcon. Ono na šta se odnosi ovaj objekat je slika "firefox.jpg".

```
import java.awt.*;
import javax.swing.*;

class ScrolledPane extends JFrame
{
   private JScrollPane scrollPane;

   public ScrolledPane() {
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setTitle("ScrolledPane");
      setSize(300, 200);
      setBackground( Color.gray );
      scrollPane = new JScrollPane();
      Icon image = new ImageIcon("firefox.jpg");
      JLabel label = new JLabel(image);
      scrollPane.getViewport().add(label);
      getContentPane().setLayout(new GridLayout(2,1));
      getContentPane().add(scrollPane);
   }

   public static void main( String args[] ) {
      ScrolledPane mainFrame  = new ScrolledPane();
      mainFrame.setVisible(true);
```

```
    }
}
```

Slika 5.51. Skrolovanje slike

Slika 5.52. Izgled aplikacije ScrolledPane

Povećanjem veličine prozora slika se može prikazati u celosti, a ako je manja pojavljuje se potreban klizač za skrolovanje.

5.2.10. Klasa JTabbedPane

U sledećem programskom kodu (slika 5.53) korišćena je klasa `JTabbedPane`. Kreiraju se tri tab-a:

- unos korisničkog imena i lozinke (maskirani unos);
- grupa programskih dugmadi;
- grupa prostora za unos teksta.

Svakom tab-u se pridružuje panel na koji će se postavljati odgovarajuće navedene komponente (`panel1, panel2, panel3`).

```
import java.awt.*;
import javax.swing.*;

class TabbedPane extends JFrame
{
   private JTabbedPane tabbedPane;
   private JPanel panel1;
   private JPanel panel2;
   private JPanel panel3;

   public TabbedPane() {
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setTitle( "JTabbedPane" );
```

```
    setSize( 300, 200 );
    setBackground( Color.gray );
    // kreiranje tab-ova
    createPage1();
    createPage2();
    createPage3();
    // kreiranje tabbed pane
    tabbedPane = new JTabbedPane();
    tabbedPane.addTab( "Page 1", panel1 );
    tabbedPane.addTab( "Page 2", panel2 );
    tabbedPane.addTab( "Page 3", panel3 );
    getContentPane().add( tabbedPane, BorderLayout.CENTER );
 }
 public void createPage1() {
    panel1 = new JPanel();
    panel1.setLayout( null );//apsolutno postavljanje
    JLabel label1 = new JLabel( "Username:" );
    label1.setBounds( 10, 15, 150, 20 );
    panel1.add( label1 );
    JTextField field = new JTextField();
    field.setBounds( 10, 35, 150, 20 );
    panel1.add( field );
    JLabel label2 = new JLabel( "Password:" );
    label2.setBounds( 10, 60, 150, 20 );
    panel1.add( label2 );
    JPasswordField fieldPass = new JPasswordField();
    fieldPass.setBounds( 10, 80, 150, 20 );
    panel1.add( fieldPass );
 }
 public void createPage2()  {
    panel2 = new JPanel();
    panel2.setLayout( new BorderLayout() );
    panel2.add( new JButton("North"), BorderLayout.NORTH );
    panel2.add( new JButton("South"), BorderLayout.SOUTH );
    panel2.add( new JButton("East" ), BorderLayout.EAST );
    panel2.add( new JButton("West" ), BorderLayout.WEST );
    panel2.add( new JButton("Center"), BorderLayout.CENTER );
 }
public void createPage3()  {
    panel3 = new JPanel();
    panel3.setLayout( new GridLayout( 3, 2 ) );
    panel3.add( new JLabel( "Field 1:" ) );
    panel3.add( new TextArea() );
    panel3.add( new JLabel( "Field 2:" ) );
    panel3.add( new TextArea() );
```

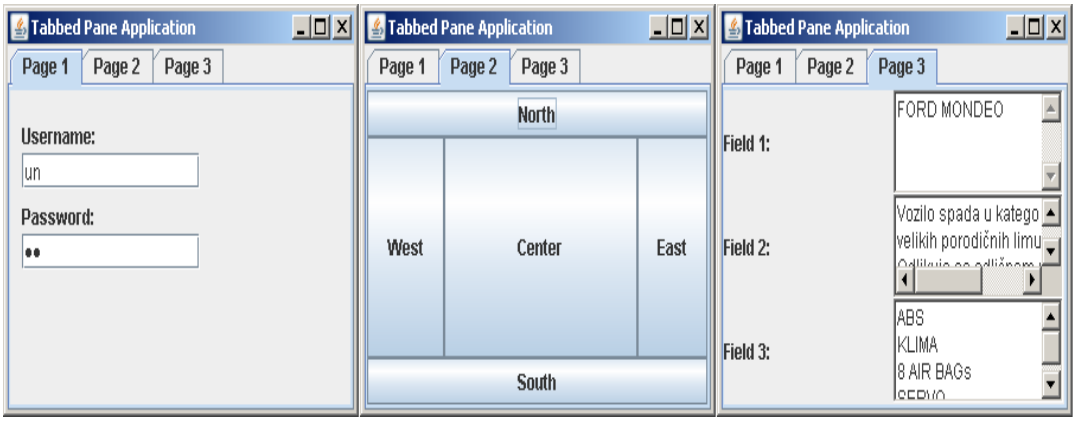
```
        panel3.add( new JLabel( "Field 3:" ) );
        panel3.add( new TextArea() );
    }

    public static void main( String args[] ) {
        TabbedPane mainFrame    = new TabbedPane();
        mainFrame.setVisible( true );
    }
}
```

Slika 5.53. Upotreba klase JTabbedPane

Prvi panel ima `null-layout` raspoređivač i sadrži labele i pripadna tekstualna polja za `Username` i `Password`. Drugi panel sadrži 5 programskih dugmadi raspoređenih u `BorderLayout`. Treći panel sadrži 3 para labele i oblasti za tekst, koji su smešteni korišćenjem `GridLayouta` u matricu 3x2. Podešavanja izgleda labela u prvom panelu se postiže metodom `setBounds` koja prihvata argumente: koordinate gornjeg levog ćoška labele, dužina labele te visina labele. Za lozinku se koristi element `JPasswordField` koji prikazuje kružiće imesto ukucanih karaktera. U drugom panelu se dodavanje programske dugmadi obavlja metodom add koja prihvata dva argumenta: referencu na objekat koji se dodaje u panel i poziciju na koju se smešta u panel.

U `tabbedpane` (`JTabbedPane`) dodaju se tri taba metodom `addTab` koja prihvata dva argumenta: naziv taba i panel koji tab sadrži (slika 5.54). U `JFrame` se na kraju dodaje `tabbedpane`.



Slika 5.54. Izgled aplikacije TabbedPane

5.3. JavaFX

JavaFX predstavlja čist Java API u smislu da se ne oslanja na prethodnu tehnologiju (`Swing` se oslanjao na `awt`). Ideja je bila da se napravi alat za GUI razvoj Internet aplikacija (*RIA – Rich Internet Applications*).

JavaFX je Java biblioteka predviđena za razne tipove uređaja (mobilni telefoni, pametni telefoni, računari, tablet računari itd) pri čemu su obuhvaćeni: audio, video, animacija i grafika.

Paradigma JavaFXa je analogna pozorištu:

- postoji pozornica (javafx.stage.Stage);
- na pozornicu se može postaviti scena (javafx.scene.Scene);
- scena ima svoj raspoređivač komponenti;
- na scenu se postavljaju komponente.

JavaFX aplikacija proširuje klasu javafx.application.Application. Startovanje JavaFX aplikacije kreće kada se iz metode main pozove metoda `launch(String[])` sa argumentima iz komandne linije. Nakon ovoga će se automatski pozvati metoda `start(Stage)` čiji je parametar referenca na objekat klase javafx.stage.Stage (dobijena je pozornica).

Graf scene je drvo sa korenim čvorom (root) te čvorovima-listovima i čvorovima-granama.

Klasa `javafx.scene.Node` je bazna klasa za predstavljanje čvorova grafa scene. Najčešće se koriste čvorovi kontrole ulaza/izlaza ili Shape objekti.

Klasa `javafx.scene.Parent` je bazna klasa za čvorove grafa scene koji mogu imati decu koja upravlja svim hijerarhijskim operacijama grafa scene (dodavanje/uklanjanje dece čvorova...)

Čvorovi-grane su tipa `javafx.scene.Parent` čije su podklase:

- `Group`,
- `Region`,
- `Control`.

Čvorovi-listovi su specifično tipa: `Rectangle`, `Text`, `ImageView`, `MediaView`, ili bilo koje klase koja nema decu.

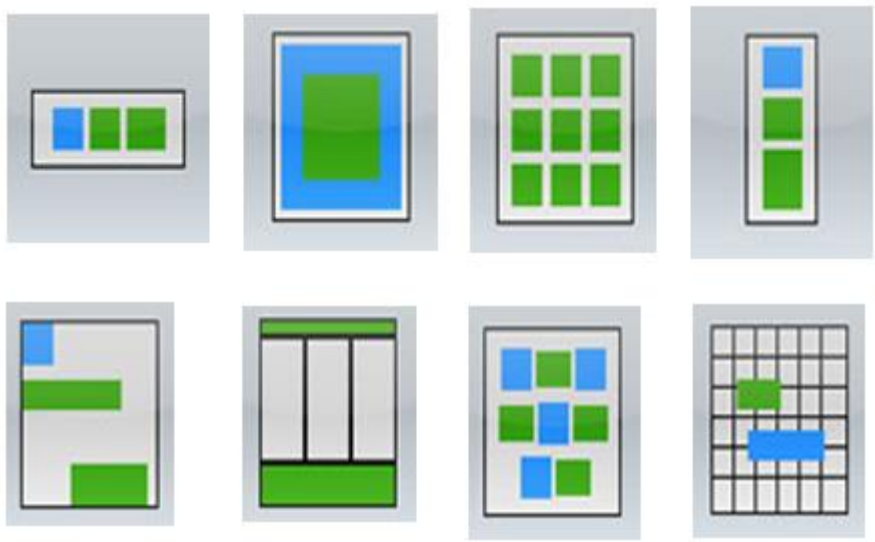
Nakon dodavanja čvorova, scena se postavlja na pozornicu. Korišćenjem metoda show (kao i ranije) prikazuje se JavaFX prozor.

JavaFX sadrži niz raspoređivačkih (layout) klasa (u paketu

`javafx.scene.layout`) koje na različite načine raspoređuju komponente na sceni (slika 5.55):

- `AnchorPane`
- `BorderPane`
- `FlowPane`
- `GridPane`
- `HBox`
- `StackPane`
- `TilePane`
- `VBox`

Raspoređivačke klase određuju poziciju i veličinu svih čvorova na sceni tako da se menjanjem veličine prozora automatski menjaju veličine čvorova.



Slika 5.55 Izgled raspoređenih komponenti s leva na desno ododzgo na dole: a) Hbox, b) StackPane, c) TilePane, d) Vbox, e) AnchorPane, f) BorderPane, g) FlowPane, h) GridPane

5.3.1. Layout klase

Layout klasa `HBox` čvorove-decu raspoređuje u red (horizontalno). Veličina čvorova dece se povećava do njihove željene širine (preferred widths), ali postoji mogućnost da se zada da se pojedini čvorovi-deca povećavaju do njihove maksimalne zadate širine.

Layout klasa `VBox` čvorove-decu raspoređuje u kolonu (vertikalno). Svojstvo Padding upravlja razmakom između čvorova. Moguće je podešavanje margina oko čvorova.

Layout klasa `FlowPane` smešta čvorove-decu u redove tako što nakon

popunjenog jednog reda prelazi u drugi. Podrazumevana pozicija reda sa komponentama je centar kontejnera, a podrazumevana orijentacija je sa leva na desno. Može se zadati podrazumevani razmak među komponentama. Veličina čvorova-dece se ne menja. Može se zadati i orijentacija.

Konstruktori klase `FlowPane`:

```
FlowPane()
```

kreira horizontalni `FlowPane` sa horizontalnim i vertikalnim razmakom 0 (hgap/vgap = 0).

```
FlowPane(double hgap, double vgap)
```

kreira horizontalni `FlowPane` sa zadatim hgap i vgap.

```
FlowPane(Orientation orientation)
```

kreira `FlowPane` zadate orijentacije sa horizontalnim i vertikalnim razmakom 0 (hgap/vgap = 0)

```
FlowPane(Orientation orientation, double hgap, double vgap)
```

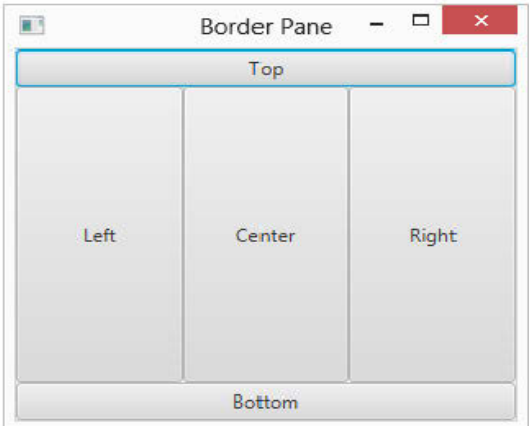
kreira `FlowPane` zadate orijentacije sa zadatim hgap i vgap.

```
Image images[] = { ... };
FlowPane flow = new FlowPane(); //horizontalni FlowPane
flow.setVgap(8);
flow.setHgap(4);
flow.setPrefWrapLength(300); // preferred width = 300
for (int i = 0; i < images.length; i++) {
  flow.getChildren().add(new ImageView(images[i]);
}
FlowPane flow = new FlowPane(Orientation.VERTICAL);
flow.setColumnHalignment(HPos.LEFT); // levo poravnanje elemenata
flow.setPrefWrapLength(200);         // visina = 200
for (int i = 0; i < titles.size(); i++) {
  flow.getChildren().add(new Label(titles[i]);
}
```

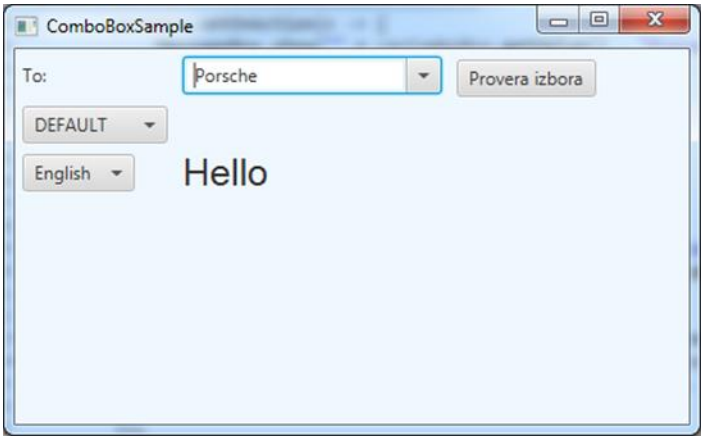
`BorderPane` raspoređuje komponente (`top`, `left`, `center`, `right`, `bottom`) tako da kontejner bude popunjen (slika 5.56). Bilo koja pozicija može da bude `null`. Pozadina i ivice mogu se stilizovati pomoću CSS.

```
BorderPane borderpane = new BorderPane();
ToolBar toolbar = new ToolBar();
HBox statusbar  = new HBox();
Region reg = new Region();
borderPane.setTop(toolbar);
```

```
borderPane.setCenter(reg);
borderPane.setBottom(statusbar);
```



Slika 5.56. Primer BorderPane rasporeda



Slika 5.57. Primer GridPane rasporeda

GridPane raspoređuje čvorove u fleksibilnu pravougaonu mrežu (slika 5.57). Čvorovi mogu da budu smešteni u bilo koje ćelije u mreži.

```
GridPane gridpane = new GridPane();
Button button = new Button();
GridPane.setColumnIndex(button, 2);
GridPane.setRowIndex   (button, 1);
Label label = new Label();
GridPane.setConstraints(label, 3, 1); //kolona pa red
gridpane.getChildren().addAll(button, label);
gridpane.add(new Button(), 2, 1);
//postavljanje elementa u kolonu i red
```

```
gridpane.add(new Label(), 3, 1);
```

Metoda `add` klase `GridPane` ima redom sledeće parametre:

- node, element koji se raspoređuje
- column, kolona u koju se element raspoređuje
- row, red u koji se element raspoređuje
- columnSpan, pokazuje koliko element zauzima kolona
- rowSpan, pokazuje koliko element zauzima redova.

5.3.2. Paket javafx.geometry

Paket `javafx.geometry` obezbeđuje skup 2D klasa za definisanje i obavljanje operacija nad objektima u dvodimenzionalnoj geometriji (npr. `Insets`, `Point2d`, `Rectangle2d`, ...).

Slede značajne klase u paketu `javafx.geometry`:

- `javafx.geometry.Pos`
  - Vertikalno i horizontalno pozicioniranje i poravnanje.
  - Skup vrednosti za opisivanje vertikalnog i horizontalnog pozicioniranja i poravnanja kao što sledi:
    - `TOP_LEFT, TOP_CENTER, TOP_RIGHT, CENTER_LEFT, CENTER, CENTER_RIGHT, BASELINE_LEFT,BASELINE_CENTER, BASELINE_RIGHT, BOTTOM_LEFT, BOTTOM_CENTER, BOTTOM_RIGHT.`
  - Metode:
    - `HPos getHpos(),` vraća horizontalnu poziciju/poravnanje
    - `VPos getVpos(),` vraća vertikalnu poziciju/poravnanje
    - `static Pos valueOf(java.lang.String name),` vraća enum konstantu zadatog imenom name
    - `static Pos[] values(),` vraća niz konstanti enum tipa, redom kojim su deklarisani.
- `javafx.geometry.Hpos`
  - Horizontalno pozicioniranje i poravnanje.
  - Skup vrednosti za opisivanje horizontalnog pozicioniranja i poravnavanja:
    - LEFT, CENTER, RIGHT
  - Metode:
    - `static HPos valueOf(java.lang.String name)` vraća enum konstantu zadatog imenom name

        - `static HPos[] values()`
          vraća niz konstanti enum tipa, redom kojim su deklarisani.

- `javafx.geometry.Vpos`
  - Vertikalno pozicioniranje i poravnanje.
  - Skup vrednosti za opisivanje vertikalnog pozicioniranja i poravnanja.
    - TOP, CENTER, BASELINE, BOTTOM
  - Metode:
    - `static VPos valueOf(java.lang.String name)`
      vraća enum konstantu zadatog imenom name
    - `static VPos[] values()`
      vraća niz konstanti enum tipa, redom kojim su deklarisani.

- `javafx.geometry.HorizontalDirection`
  - Postavljanje horizontalnog smera.
  - Vrednosti ka levo LEFT, ka desno RIGHT.
  - Metode:
    - `static HorizontalDirection valueOf(String name)`
      vraća enum konstantu zadatog imenom name.
    - `static HorizontalDirection[] values()`
      vraća niz konstanti enum tipa, redom kojim su deklarisani.

- `javafx.geometry.VerticalDirection`
  - Postavljanje vertikalnog smera.
  - Vrednosti ka dole DOWN, ka gore UP.
  - Metode:
    - `static VerticalDirection valueOf(String name)`
      vraća enum konstantu zadatog imenom name.
    - `static VerticalDirection[] values()`
      vraća niz konstanti enum tipa, redom kojim su deklarisani.

- `javafx.geometry.Side`
  - Određivanje strane pravougaonika (naslova dijagrama, brojnih osa,...)
  - Skup vrednosti za opisivanje strane: BOTTOM, LEFT, RIGHT, TOP
  - Metode:
    - `boolean isHorizontal()`
      da li je horizontalna strana (true za TOP i BOTTOM)
    - `boolean isVertical()`
      da li je vertikalan strana (true za LEFT i RIGHT).

- `static Side valueOf(java.lang.String name)`
  vraća enum konstantu zadatog imenom name.
- `static Side[] values()`
  vraća niz konstanti enum tipa, redom kojim su deklarisani..

– `javafx.geometry.Orientation`
  - Određivanje horizontalne i vertikalne orijentacije.
  - Skup vrednosti za određivanje horizontalne i vertikalne orijentacije:
    - `HORIZONTAL`, horizontalna (levo, desno) orijentacija.
    - `VERTICAL`, vertikalna (vrh, dno) orijentacija.
  - Metode:
    - `static Orientation valueOf(java.lang.String name)`
      vraća enum konstantu zadatog imenom name
    - `static Orientation[] values()`
      vraća niz konstanti enum tipa, redom kojim su deklarisani.

– `javafx.geometry.Insets`
  - Postavljanje rastojanja za svaku stranu elementa koji se prikazuje.
  - Konstruktori:
    - `Insets(double topRightBottomLeft)`
      Formira se Insets objekat koji ima jednaku vrednost rastojanja za sva četiri razmaka.
    - `Insets(double top, double right, double bottom, double left)`
      Formira se Insets objekat kome su redom (gore, desno, dole, levo) definisane vrednost rastojanja za sva četiri razmaka.

– `javafx.scene.paint.Color`
  - Klasa za boje u javafx.
  - Standardni konstruktor:
    - `Color(double red, double green, double blue, double opacity)`
      red, green, blue, opacity (sve u intervalu [0,1])
  - U RGB modelu boja se definiše kao kombinacija tri boje (RGB, odnosno, crvene, zelene i plave) u interval 0-255.
    - `Color boja = Color.rgb(0,0,0);` crna
    - `Color boja = Color.rgb(255,255,255);` bela

- U HSB modelu boja je definisana kao kombinacija nijanse, zasićenosti, osvetljenja i neprozirnosti:
  - hsb(double hue, double saturation, double brightness)
  - hsb(double hue, double saturation, double brightness, double opacity)
  - `Color boja = Color.hsb(270,1.0,1.0);`
  - `Color boja = Color.hsb(270,1.0,1.0,1.0);`
- Boja ima podrazumevano alpha vrednost 1.0 (ili 255 prema opsegu) što znači neprozirnost (0 je potpuna providnost).
  - `Color boja = new Color(0,0,1,0.7);`
    `// alpha = 0.7`
- RGB boja može se postaviti metodom web:
  - `web(java.lang.String colorString)`
  - `web(java.lang.String colorString, double opacity)`
  - `Color boja = Color.web("0x0000FF",1.0);`
    `//neprovidna plava`
- Klasa Color definiše statičke konstante za boje: WHITE (255,255,255), PINK (255,175,175), ORANGE (255,200,0), GRAY (128,128,128), CYAN (0,255,255), GREEN (0,255,0), BLACK (0,0,0), RED (255,0,0), LIGHTGRAY (192,192,192), MAGENTA (255,0,255), YELLOW (255,255,0), DARKGRAY (64,64,64), BLUE (0,0,255).
- Kreirana boja se može posvetliti ili potamniti respektivno metodama:
  - `boja.brighter();`
  - `boja.darker();`
- Vraćanje komponenti boje postiže se metodama:
  - `getRed();`
  - `getGreen();`
  - `getBlue();`
- Poređenje boja vrši se metodom:
  - `equals();`

– `javafx.scene.Cursor`
- Definiše statičke konstante koje određuju tip kursora:
  - `CLOSED_HAND, CROSSHAIR, DEFAULT, HAND, TEXT, WAIT`

5.3.3. JavaFX, upravaljanje događajima

Sledi prikaz koda koji prikazuje programsko dugme i gde postoji reakcija na događaj klik mišem na programsko dugme. Izgled aplikacije dat je na slici 5.3.4.

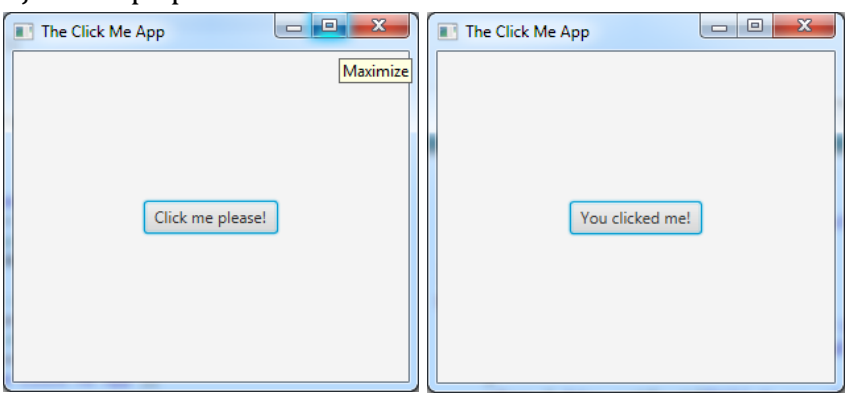
U main-u klase `FXClickMe` koja proširuje klasu `javafx.application` je startovana JavaFX aplikacija (sinhronom metodom `launch(String[])`) koja će automatski pozvati metodu `start(Stage)`. Metoda `start` prihvata kao argument 'pozornicu' (`Stage`) koja se koristi da bi se na nju postavila scena.

Programsko dugme `btn` metodom `setText("Click me please!")` postavlja tekst koji će biti ispisan na tom programskom dugmetu. Metoda programskog dugmeta `setOnAction` postavlja koja metoda klase `FXClickMe` obrađuje događaj klika na to programsko dugme. Parametar ove metode je napisan kao lambda izraz `(Beograd -> buttonClick())`. `Beograd` je `ActionEvent` referenca koju u ovom primeru nećemo koristiti u kodu obrađivača.

Metoda `buttonClick()` klase `FXClickMe` biće pozvana kada se desi klik na programsko dugme btn. U ovoj metodi se tekst ispisan na programskom dugmetu dohvata metodom `getText()` te se proverava šta je trenutno ispisano na programskom dugmetu, a onda na osnovu toga postavlja novi ispis metodom `setText("novi tekst")`.

Raspoređivač `pane` tipa `BorderPane` komponentu `btn` tipa `Button` postavlja na centar pozivom metode `setCenter(btn)`.

Sada se kreira scena `scene` koja prikazuje `pane` u prozoru veličine 300x250 piksela. Ova scena se postavlja na pozornicu `primaryStage` pozivom metode pozornice `setScene(scene)`. Naslov prozora se postavlja metodom pozornice `setTitle("The Click Me App")`, a onda sledi poziv metode pozornice `show()` koja prikazuje scenu. Videti listing na slici 5.59. Svi potrebni importi su navedeni u ovom i listinzima koji slede. Iako razvojno okruženje omogućuje pomoć za import klasa, biće navedeni svi importi zbog lakšeg razumevanja kom paketu koja klasa pripada.



Listing 5.58 Izgled aplikacije obrade klika na programsko dugme

```
import javafx.application.*;
import javafx.stage.*;
import javafx.scene.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
public class FXClickMe extends Application {
    public static void main(String[] args) {
        launch(args); //sinhrona
    }
    Button btn;
    @Override public void start(Stage primaryStage) {
        // Create the button
        btn = new Button();
        btn.setText("Click me please!");
        btn.setOnAction(Beograd -> buttonClick());
        // Add the button to a layout pane
        BorderPane pane = new BorderPane();
        pane.setCenter(btn);
        // Add the layout pane to a scene
        Scene scene = new Scene(pane, 300, 250);
        // Finalize and show the stage
        primaryStage.setScene(scene);
        primaryStage.setTitle("The Click Me App");
        primaryStage.show();
    }
    public void buttonClick() {
        if (btn.getText() == "Click me please!") {
               btn.setText("You clicked me!");
        }
        else { btn.setText("Click me please!");
        }
    }
}
```

Slika 5.59. Kod obrade klika na programsko dugme

U sledećem primeru glavna klasa `AddSubstract` proširuje klasu `Application` i implementira interfejs `EventHandler<ActionEvent>`. U metodi `main` pozvana je sinhrona metoda `launch` kao i u prethodnom primeru, tako da opet metoda `start` prihvata referencu na pozornicu.

Kreiraju se dva programska dugmeta `btnAdd` i `btnSubstract` pri čemu klik mišem na programsko dugme `btnAdd` inkrementira vrednost brojača (`iCounter`), dok klik mišem na programsko dugme `btnSubstract` smanjuje vrednost navedenog brojača. Vrednost brojača se prikazuje na labeli `lbl` pozivom metode labele `setText()` kojoj se prosleđuje string reprezentacija vrednosti brojača (`Integer.toString(iCounter)`).

Metoda programskog dugmeta `setText("TEKST")` dodeljuje tekst `TEKST` koji će biti ispisan na programskom dugmetu (u primeru `Add` i `Substract` respektivno za dugmad `btnAdd`, `btnSubstract`). Metoda programskog dugmeta `setOnAction(this)` dodeljuje obrađivač događaja koji je implementiran u klasi `AddSubstract`. U našem primeru isti je obrađivač za oba programska dugmeta.

Raspoređivač komponenti pane je tipa `Hbox(10)` čime je zadato da je horizontalni razmak (spacing) između komponenti koje se raspoređuju 10 piksela. `Hbox` klasa metodom `getChildren().addAll(lbl, btnAdd, btnSubtract)` uzima prethodno stanje `Hbox` objekta i dodaje mu nove elemente (ovde su to labela i dva programska dugmeta).

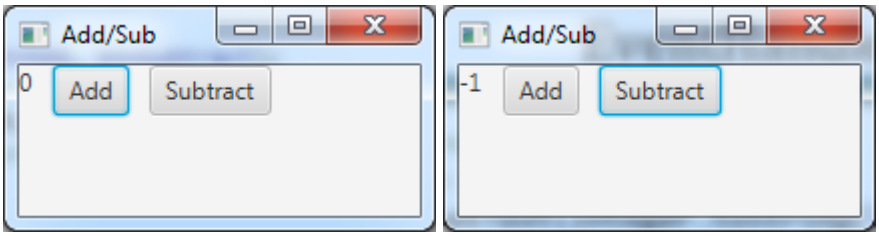
Kreira se scena `scene` koja prikazuje `pane` u prozoru veličine 300x250 piksela. Ova scena se postavlja na pozornicu `primaryStage` pozivom metode pozornice `setScene(scene)`. Naslov prozora se postavlja metodom pozornice `setTitle("Add/Subb")`, a onda sledi poziv metode pozornice `show()` koja prikazuje scenu.

Metoda `handle` interfejsa `EventHandler<ActionEvent>` obrađuje klik na programsko dugme `btnAdd` ili `btnSubstract` tako da se pozivom metode `getSource()` objekta e klase `ActionEvent` proveri o kojoj je referenci (ovde programskom dugmetu) reč. Ako je klik bio na programsko dugme `btnAdd` sledi inkrementiranje vrednosti brojača `iCounter` i prikaz njegove nove vrednosti na labeli `lbl` pozivom labeline metode `setText(Integer.toString(iCounter))`. U slučaju da je klik bio na programsko dugme `btnSubstract` dekrementira se vrednost brojača `iCounter`, a onda prikaže nova vrednost brojača `iCounter` na labeli `lbl`. Videti listing na slici 5.60. Izgled aplikacije je dat na slici 5.61.

```
import javafx.application.*; import javafx.stage.*;
import javafx.scene.*; import javafx.scene.layout.*;
import javafx.scene.control.*; import javafx.event.*;
public class AddSubtract extends Application
                         implements EventHandler<ActionEvent> {
  Button btnAdd;
  Button btnSubtract;
  Label lbl;
  int iCounter = 0;
  public static void main(String[] args) {    launch(args);  }
  @Override
  public void start(Stage primaryStage) {
    // Create the Add button
    btnAdd = new Button();
    btnAdd.setText("Add");
    btnAdd.setOnAction(this);
    // Create the Subtract button
    btnSubtract = new Button();
```

```
    btnSubtract.setText("Subtract");
    btnSubtract.setOnAction(this);
    // Create the Label
    lbl = new Label();
    lbl.setText(Integer.toString(iCounter));
    // Add the buttons and label to an HBox pane
    HBox pane = new HBox(10); //spacing 10px
    pane.getChildren().addAll(lbl, btnAdd, btnSubtract);
    // Add the layout pane to a scene
    Scene scene = new Scene(pane, 200, 75);
    // Add the scene to the stage, set the title and show the
stage
    primaryStage.setScene(scene);
    primaryStage.setTitle("Add/Sub");
    primaryStage.show();
  }
  @Override
  public void handle(ActionEvent e) {
    if (e.getSource() == btnAdd) { iCounter++; }
    else {
      if (e.getSource() == btnSubtract) { iCounter--; }
    }
    lbl.setText(Integer.toString(iCounter));
  }
}
```

Slika 5.60. Kod za primer inkrementiranja/dekrementiranja brojača gde glavna klasa implementira interfejs EventHandler<ActionEvent>



Slika 5.61. Izgled aplikacije za inkrementiranje/dekrementiranje brojača

Posmatra se rešenje gde, za razliku od prethodnog primera, unutrašnja klasa implementira interfejs `EventHandler<ActionEvent>`. Videti listing na slici 5.62. Oba programska dumeta (`btnAdd, btnSubstract`) pozivaju svoju metodu `setOnAction(ch)` gde je `ch` objekat klase `ClickHandler`. Unutrašnja klasa klase `AddSubstract2` je klasa `ClickHandler` koja implementira interfejs `EventHandler<ActionEvent>` definisanjem metode `handle(ActionEvent e)`. Sada se u ovoj metodi vrši obrada događaja kao u prethodnom primeru. Ovo rešenje i prethodno rešenje su funkcionalno ekvivalentni.

```
import javafx.application.*; import javafx.stage.*;
import javafx.scene.*; import javafx.scene.layout.*;
import javafx.scene.control.*; import javafx.event.*;
public class AddSubtract2 extends Application {
  Button btnAdd; Button btnSubtract;
  Label lbl; int iCounter = 0;
  public static void main(String[] args) { launch(args); }
  @Override
  public void start(Stage primaryStage) {
    // Create a ClickHandler instance
    ClickHandler ch = new ClickHandler();
    btnAdd = new Button();      btnAdd.setText("Add");
    btnAdd.setOnAction(ch);
    btnSubtract = new Button();
    btnSubtract.setText("Subtract");
    btnSubtract.setOnAction(ch);
    lbl = new Label();
    lbl.setText(Integer.toString(iCounter));
    HBox pane = new HBox(10);
    pane.getChildren().addAll(lbl, btnAdd, btnSubtract);
    Scene scene = new Scene(pane, 200, 75);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Add/Sub");
    primaryStage.show();
  }
  private class ClickHandler implements
                                  EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
      if (e.getSource() == btnAdd) { iCounter++; }
      else {
        if (e.getSource() == btnSubtract) { iCounter--; }
      }
      lbl.setText(Integer.toString(iCounter));
    }
  }
}
```

Slika 5.62. Kod za primer inkrementiranja/dekrementiranja brojača gde unutrašnja klasa implementira interfejs EventHandler<ActionEvent>

Prikazuje se rešenje koje koristi anonimnu klasu za implementaciju interfejsa `EventHandler<ActionEvent>` za događaj klika na programsko dugme na primeru inkrementiranja i dekrementiranja brojača. Videti listing na slici 5.63. Pri pozivu metode programskog dugmeta `setOnAction` postavlja se kao argument implementacija neiemnovanog objekta klase koja implementira

interfejs `EventHandler<ActionEvent>` definišući šta njegova metoda `handle(ActionEvent e)` radi (ako je izvor događaja programsko dugme `btnAdd`, onda se brojač inkrementira i ispisuje na labeli `lbl`, odnosno, ako je izvor događaja programsko dugme `btnSubstract`, onda se brojač inkrementira i ispisuje na labeli `lbl`).

```
import javafx.application.*; import javafx.stage.*;
import javafx.scene.*; import javafx.scene.layout.*;
import javafx.scene.control.*; import javafx.event.*;
public class AddSubtract3 extends Application {
  Button btnAdd; Button btnSubtract;
  Label lbl; int iCounter = 0;
  public static void main(String[] args) { launch(args); }
  @Override
  public void start(Stage primaryStage) {
    btnAdd = new Button();
    btnAdd.setText("Add");
    //anonimna klasa
    btnAdd.setOnAction(new EventHandler<ActionEvent>() {
      public void handle(ActionEvent e) {
                    iCounter++;
                    lbl.setText(Integer.toString(iCounter));
      }
    });
    btnSubtract = new Button();
    btnSubtract.setText("Subtract");
    btnSubtract.setOnAction(new EventHandler<ActionEvent>(){
      public void handle(ActionEvent e) {
                    iCounter--;
                    lbl.setText(Integer.toString(iCounter));
      }
    });
    lbl = new Label();
    lbl.setText(Integer.toString(iCounter));
    HBox pane = new HBox(10);
    pane.getChildren().addAll(lbl, btnAdd, btnSubtract);
    Scene scene = new Scene(pane, 200, 75);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Add/Sub");
    primaryStage.show();
  }
}
```

Slika 5.63. Kod za primer inkrementiranja/dekrementiranja brojača gde anonimna klasa implementira interfejs EventHandler<ActionEvent>

5.3.3.1. Korišćenje lambda izraza

Sada se posmatra rešenje inkrementiranja/dekrementiranja brojača gde se koristi kompaktniji kod u vidu lambda izraza. Interfejs `EventHandler` ima samo jednu apstraktnu metodu (metoda `handle`) tako da je `EventHandler` funkcionalni interfejs i može se koristiti korišćenjem lambda izraza (*lambda expression*). U ovom slučaju korišćenjem lambda izraza kreira se anonimna klasa koja implementira funkcionalni interfejs tako što se definiše samo: parametar i telo metode. Java kompajler sada zaključuje ostatak koda na temelju konteksta u kom se koristi lambda izraz. Parametar i telo metode se razdvojeni znakom -> kao što sledi:

```
e ->
  {
     iCounter++;
     lbl.setText(Integer.toString(iCounter);
  }
```

Ovde lambda izraz implementira funkcionalni interfejs čija jedina metoda prihvata jedini parametar označen sa `e` (tako da može i bez malih zagrada). U telu metode se inkrementira `iCounter` i ažurira tekst labele `lbl` da prikaže novu vrednost promenljive `iCounter` .

Registrovanje lambda izraza kao upravljača događaja za programsko dugme je kao što sledi:

```
btnAdd.setOnAction( e ->
                {
                   iCounter++;
                   lbl.setText(Integer.toString(iCounter));
                }
);
```

Naziv metode se ne mora znati pošto funkcionalni interfejs `EventHandler` ima jedan apstraktni metod (`handle`), a ne mora se znati ni naziv interfejsa pošto je interfejs određen kontekstom (`setOnAction` metoda ima jedan parametar tipa `EventHandler<ActionEvent>`) tako da će sve da odradi kompajler. Pripadni "kompaktni" kod za inkrementiranje/dekrementiranje sa lambda izrazom je dat na listingu 5.64.

```
import javafx.application.*; import javafx.stage.*;
import javafx.scene.*; import javafx.scene.layout.*;
import javafx.scene.control.*;
public class AddSubtract4 extends Application {
    Button btnAdd; Button btnSubtract; Label lbl;
    int iCounter = 0;
    public static void main(String[] args) {launch(args);  }
```

```
    @Override
    public void start(Stage primaryStage) {
        btnAdd = new Button();
        btnAdd.setText("Add");
        btnAdd.setOnAction( e ->
            {
                iCounter++;
                lbl.setText(Integer.toString(iCounter));
            } );
        btnSubtract = new Button();
        btnSubtract.setText("Subtract");
        btnSubtract.setOnAction( e ->
            {
                iCounter--;
                lbl.setText(Integer.toString(iCounter));
            } );
        lbl = new Label();
        lbl.setText(Integer.toString(iCounter));
        HBox pane = new HBox(10);
        pane.getChildren().addAll(lbl, btnAdd, btnSubtract);
        Scene scene = new Scene(pane, 200, 75);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Add/Sub");
        primaryStage.show();
    }
}
```

Slika 5.64. Inkrementiranja/dekrementiranja brojača rešen korisšćenjem lambda izraza

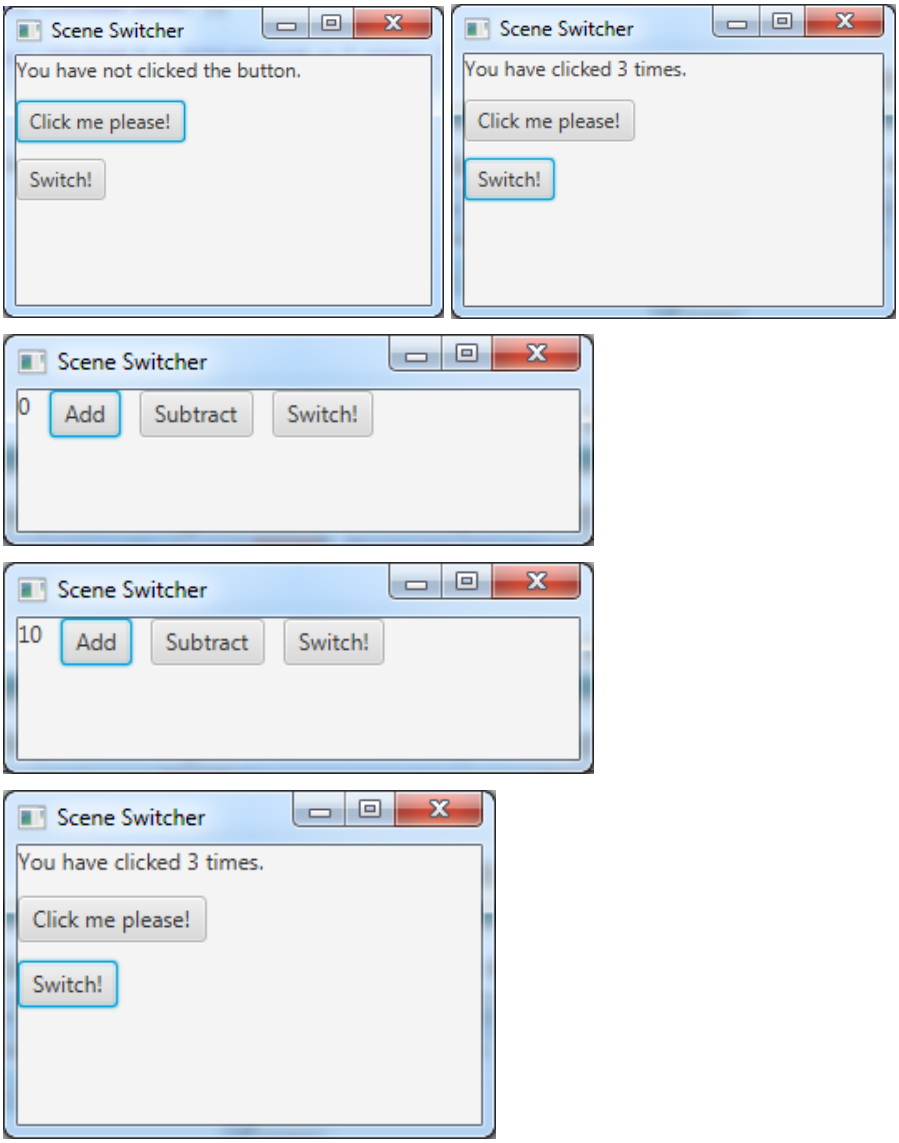
Razmislite o rešenju inkremetiranja/dekrementiranja u kome bi bio implementiran poziv korisničkih metoda korišćenjem lambda izraza:

```
...
btnAdd.setOnAction( e -> btnAdd_Click() );
btnSubtract.setOnAction( e -> btnSubtract_Click() );
...
private void btnAdd_Click()
{
     iCounter++;
     lbl.setText(Integer.toString(iCounter));
}
private void btnSubtract_Click()
{
     iCounter--;
     lbl.setText(Integer.toString(iCounter));
}
...
```

te razmislite o prosleđivanju argumenta e i njegovog korišćenja u pozvanoj metodi.

5.3.3.2. Smenjivanje scena

Pozornica (`Stage`) ne mora da se veže za samo jednu scenu. Sledi primer u kome se dve scene smenjuju na istom `Stage`-u. Na slici 5.65. prikazani su izgledi aplikacije koja kombinuje primere: klik na programsko dugme kao jedna scena i inkrementiranje/dekrementiranje brojača kao druga scena. Na obe scene dodato je programsko dugme *Switch!* koje omogućuje smenjivanje navedene dve scene (klikom na *Switch!*). Programski kod je dat na slici 5.66.



Slika 5.65. Izgled aplikacije koja smenjuje dve scene

```
import javafx.application.*; import javafx.stage.*;
import javafx.scene.*; import javafx.scene.layout.*;
import javafx.scene.control.*;
public class SceneSwitcher extends Application {
  public static void main(String[] args) { launch(args); }
  // class fields for Click-Counter scene
  int iClickCount = 0; Label lblClicks;
  Button btnClickMe; Button btnSwitchToScene2;
  Scene scene1;
  // class fields for Add-Subtract scene
  int iCounter = 0; Label lblCounter;
  Button btnAdd; Button btnSubtract;
  Button btnSwitchToScene1;
  Scene scene2;
  // class field for stage
  Stage stage;
  @Override
  public void start(Stage primaryStage) {
    stage = primaryStage;
    // Build the Click-Counter scene
    lblClicks = new Label();
    lblClicks.setText("You have not clicked the button.");
    btnClickMe = new Button();
    btnClickMe.setText("Click me please!");
    btnClickMe.setOnAction(e -> btnClickMe_Click());
    btnSwitchToScene2 = new Button();
    btnSwitchToScene2.setText("Switch!");
    btnSwitchToScene2.setOnAction(e ->
                                  btnSwitchToScene2_Click());
    VBox pane1 = new VBox(10);
    pane1.getChildren().addAll(lblClicks, btnClickMe,
                                          btnSwitchToScene2);
    scene1 = new Scene(pane1, 250, 150);

    // Build the Add-Subtract scene
    lblCounter = new Label();
    lblCounter.setText(Integer.toString(iCounter));
    btnAdd = new Button();      btnAdd.setText("Add");
    btnAdd.setOnAction(e -> btnAdd_Click());
    btnSubtract = new Button();
    btnSubtract.setText("Subtract");
    btnSubtract.setOnAction(e -> btnSubtract_Click());
    btnSwitchToScene2 = new Button();
    btnSwitchToScene2.setText("Switch!");
    btnSwitchToScene2.setOnAction(e ->
                                  btnSwitchToScene1_Click());
    HBox pane2 = new HBox(10);
```

```
    pane2.getChildren().addAll(lblCounter, btnAdd,
                               btnSubtract,btnSwitchToScene2);
    scene2 = new Scene(pane2, 300, 75);

    // Set the stage with scene 1 and show the stage
    primaryStage.setScene(scene1);
    primaryStage.setTitle("Scene Switcher");
    primaryStage.show();
  }
  // Event handlers for scene 1
  public void btnClickMe_Click() {
    iClickCount++;
    if (iClickCount == 1) { lblClicks.setText("You have clicked
once.");
    } else {
       lblClicks.setText("You have clicked " + iClickCount + "
times.");
    }
  }
  private void btnSwitchToScene2_Click() {
    stage.setScene(scene2);
  }

  // Event handlers for scene 2
  private void btnAdd_Click() {
    iCounter++;
    lblCounter.setText(Integer.toString(iCounter));
  }
  private void btnSubtract_Click() {
    iCounter--;
    lblCounter.setText(Integer.toString(iCounter));
  }
  private void btnSwitchToScene1_Click() {
    stage.setScene(scene1); }
}
```

Slika 5.66. Kod za smenjivanje dve scene

U odnosu na ranije date primere dodato je u prvu scenu (`scene1`) programsko dugme `btnSwitchToScene2` (u `VBox`). Ovo programsko dugme metodom `setOnAction` i lambda izrazom kao parametrom navodi metodu `btnSwitchToScene2_Click()`. U ovoj metodi postavlja se na pozornicu druga scena (`stage.setScene(scene2)`). Na isti način dodato je programsko dugme `btnSwitchToScene1` (u `HBox`) u drugu scenu (`scene2`). Ovo programsko dugme metodom `setOnAction` i lambda izrazom kao parametrom navodi metodu `btnSwitchToScene1_Click()`. U ovoj metodi postavlja se na pozornicu prva scena (`stage.setScene(scene1)`).

5.3.3.3. Korisnička klasa MessageBox

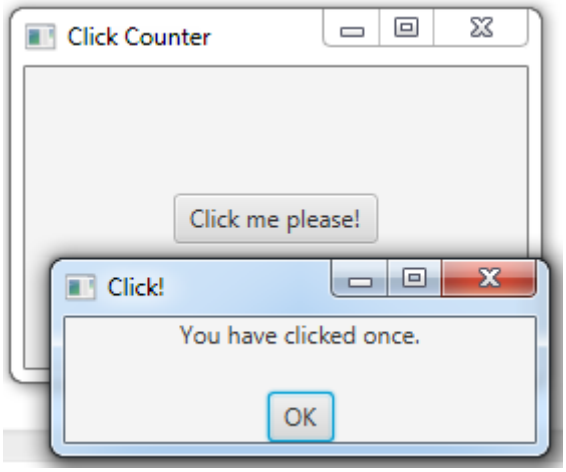
Kreira se klasa `MessageBox` koja će služiti za ispisivanje poruka kao modalni dijalog (slika 5.67). Ova klasa ima jednu metodu `show` koja kao parametre prihvata dva stringa (string poruke i string naslova). U metodi `show` kreira se pozornica `stage` kojoj se postavlja modalitet metodom `initModality(Modality.APPLICATION_MODAL)`. Ovaj modalitet blokira propagaciju događaja ka bilo kom prozoru aplikacije. Ako se želi blokiranje propagacije događaja kroz hijerahiju roditeljskih komponenti, onda se koristi opcija `Modality.WINDOW_MODAL`. U slučaju da se ne želi blokada propagacije koristi se opcija `Modality.NONE`. Pozornici se postavlja naslov i minimalna širina od 250 piksela. Kreiraju se: labela `lbl` koja će prikazati poruku i programsko dugme `btnOK` kome je dodeljena akcija zatvaranja pozornice stage. Obe komponente se centrirano postavljaju u `VBox pane` sa međurazmakom komponenti od 20 piksela. Kreira se scena `scene` koja centrirano sadrži `pane`, a koja se onda stavlja na pozornicu `stage` i na kraju se poziva sinhrona metoda pozornice `showAndWait()` koja čeka na zatvaranje pozornice.

```
import javafx.application.*;   import javafx.stage.*;
import javafx.scene.*;         import javafx.scene.layout.*;
import javafx.scene.control.*; import javafx.geometry.*;
public class MessageBox {
  public static void show(String message, String title) {
    Stage stage = new Stage();
    stage.initModality(Modality.APPLICATION_MODAL);
    stage.setTitle(title);
    stage.setMinWidth(250);
    Label lbl = new Label();
    lbl.setText(message);
    Button btnOK = new Button();
    btnOK.setText("OK");
    btnOK.setOnAction(e -> stage.close());
    VBox pane = new VBox(20);
    pane.getChildren().addAll(lbl, btnOK);
    pane.setAlignment(Pos.CENTER);
    Scene scene = new Scene(pane);
    stage.setScene(scene);
    stage.showAndWait();
  }
}
```

Slika 5.67. Kod klase MessageBox

Potrebno je kreirati aplikaciju čiji bi izgled bio kao na slici 5.68. Ideja je da se ispis broja klikova na programsko dugme u ovoj aplikaciji obavlja korišćenjem klase `MessageBox`. Pripadni kod je dat na slici 5.69. Razlika u odnosu na ranije

date primere za klik jeste da je sada ispis umesto na labeli prosleđen kao parametar metodi show klase `MessageBox`. Drugi prosleđeni parametar je string `Click!` koji predstavlja naslov pozornice koja se kreira u `show` metodi.



Slika 5.68. Izgled aplikacije koja koristi klasu MessageBox

```
import javafx.application.*; import javafx.stage.*;
import javafx.scene.*; import javafx.scene.layout.*;
import javafx.scene.control.*;
public class ClickCounter extends Application {
  public static void main(String[] args) { launch(args);   }
  int iClickCount = 0; Button btn;
  @Override
  public void start(Stage primaryStage)  {
    btn = new Button(); btn.setText("Click me please!");
    btn.setOnAction(e -> buttonClick());
    BorderPane pane = new BorderPane();
    pane.setCenter(btn);
    Scene scene = new Scene(pane, 250, 150);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Click Counter");
    primaryStage.show();
  }
  public void buttonClick() {
    iClickCount++;
    if(iClickCount==1)MessageBox.show(
                        "You have clicked once.","Click!");
    else MessageBox.show("You have clicked "+iClickCount
                                      +" times.", "Click!");
    }
}
```

Slika 5.69. Kod klase koja koristi klasu MessageBox

5.3.3.4. Korisnička klasa ConfirmationBox

Kreira se `ConfirmationBox` klasa koja služi kao dijalog u kome se odgovara na postavljeno pitanje odabiranjem odgovora da ili ne (`Yes`, `No`). Pripadni kod je dat na slici 5.70. Izgled `ConfirmationBox` aplikacije dat je na slici 5.71. Klasa `ConfirmationBox` ima metodu `show` koja prihvata četiri argumenta (reference): string poruke, string naslova, string za programsko dugme potvrde i string za programsko dugme odustajanja. Inicijalno se postavlja boolean `btnYesClicked` na `false`. Kao i u primeru za `MessageBox` klasu kreira se pozornica *stage* (postavlja joj se naslov, minimalna širina i modalitet koji blokira propagaciju događaja ka bilo kom prozoru aplikacije), labela poruke (`lbl`) i u ovom slučaju dva programska dugmeta (`btnYes`, `btnNo`). Ovim programskim dugmadima lambda izrazom (u metodi `setOnAction`) dodeljene su metode za obradu događaja klik: `btnYes_Clicked()`, `btnNo_Clicked()` za programsku dugmad `btnYes`, `btnNo`, respektivno. Oba dugmeta su postaljena u `HBox paneBtn`, a onda su labela `lbl` i `paneBtn` postavljeni u `VBox pane`. Kreira se scena `scene` na koju se postavlja `pane`. Scena se postavlja na pozornicu `stage`, a onda poziva sinhrona metoda pozornice `showAndWait()`. U metodama za obradu događaja klik za oba programska dugmeta zatvara se pozornica i postavlja vrednost promenljive `btnYesClicked` prema kliku na dato programsko dugme (`true` za `btnYes`). Pošto je pozornica zatvorena izvršava se naredba `return btnYesClicked` čime se prosleđuje odgovor aplikaciji koja je pozvala `show` metodu `ConfirmationBox` klase.

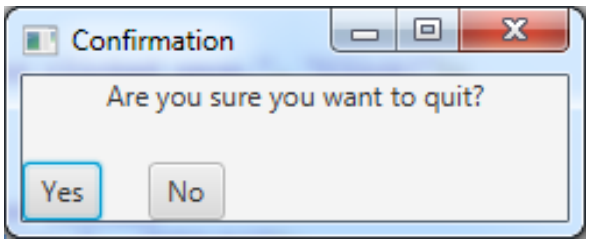
```
import javafx.application.*;  import javafx.stage.*;
import javafx.scene.*;        import javafx.scene.layout.*;
import javafx.scene.control.*;import javafx.geometry.*;

public class ConfirmationBox {
  static Stage stage;
  static boolean btnYesClicked;

  public static boolean show(String message, String title,
                             String textYes, String textNo) {
    btnYesClicked = false;
    stage = new Stage();
    stage.initModality(Modality.APPLICATION_MODAL);
    stage.setTitle(title);
    stage.setMinWidth(250);
    Label lbl = new Label();
    lbl.setText(message);
    Button btnYes = new Button();
    btnYes.setText(textYes);
    btnYes.setOnAction(e -> btnYes_Clicked());
    Button btnNo = new Button();
    btnNo.setText(textNo);
```

```
    btnNo.setOnAction(e -> btnNo_Clicked());
    HBox paneBtn = new HBox(20);
    paneBtn.getChildren().addAll(btnYes, btnNo);
    VBox pane = new VBox(20);
    pane.getChildren().addAll(lbl, paneBtn);
    pane.setAlignment(Pos.CENTER);
    Scene scene = new Scene(pane);
    stage.setScene(scene);     stage.showAndWait();
    return btnYesClicked;
  }
  private static void btnYes_Clicked() {
     stage.close();
     btnYesClicked = true;
  }
  private static void btnNo_Clicked() {
     stage.close();
     btnYesClicked = false;
  }
}
```

Slika 5.70. Kod klase ConfirmationBox



Slika 5.71. Prozor ConfirmationBox klase

Razmislite o korišćenju klase `ConfirmationBox`-a gde bi poziv bio npr:

```
  public void btnClose_Click()   {
     boolean confirm = false;
     confirm = ConfirmationBox.show(
        "Are you sure you want to quit?", "Confirmation",
        "Yes", "No");
     if (confirm) stage.close();
  }
```

5.3.3.5. Klasa TextField

Za unos teksta u JavaFX tehnologiji koristi se klasa `TextField`. Sledi kod (slika 5.72) koji omogućuje upis uloge i upis imena glumca, a onda poziva metodu `show` ranije kreirane klase `MessageBox` kojoj prosleđuje informacije preuzete iz polja za tekst.

U metodi `start` kreiraju se dve labele `lblCharacter` i `lblActor` minimalne širine 100 piksela (`setMinWidth(100)`) i desnim poravnanjem (`setAlignment(Pos.BOTTOM_RIGHT)`) te dva tekstualna polja `txtCharacter` i `txtActor`, oba širine 200 piksela (postavljanjem iste minimalne i maksimalne širine na 200 piksela (`setMinWidth(200), setMaxWidth(200)`). Inicijalni tekst za navedena tekst polja je postavljen metodom tekst polja `setPromptText` kome se prosleđuje string koji će biti ispisan sivim tekstom u datom tekst polju kada ništa nije uneseno u tekst polje.

Kreira se i programsko dugme `btnOK` minimalne širine 75 piksela čiji događaj klik obrađuje metoda `btnOK_Click()`. Ova metoda proverava da li su oba tekst polja popunjena i u slučaju da ta validacija ne prođe šalje se metodi `show` klase `MessageBox` informacija o nepotpunim podacima unosa. Ako je validacija prošla (popunjena su oba tekst polja), onda se vrši formiranje poruke konkatenacijom stringova tako da poruka sadrži informacije koja je uloga u pitanju i ko je glumac te uloge, a onda se šalje metodi `show` klase `MessageBox`.

Parovi labela i tekst polje se dodaju u pripadni `HBox` sa definisanim okolnim odstojanjem od 10 piksela. Takođe se i programsko dugme dodaje u `HBox` sa donjim desnim poravnanjem. Sva tri navedena `HBox` elementa dodaju se u `VBox` (sa definisanim razmakom među elementima od 10 piksela). Kreira se scena `scene` koja sadrži ovaj `VBox`. Scena se postavlja na pozornicu kojoj se postavlja i naslov, a onda poziva i metoda `show` za prikaz pozornice. Izgled aplikacije dat je na slici 5.73.

```
import javafx.application.*;
import javafx.stage.*; import javafx.scene.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
import javafx.geometry.*;

public class RolePlayer extends Application{

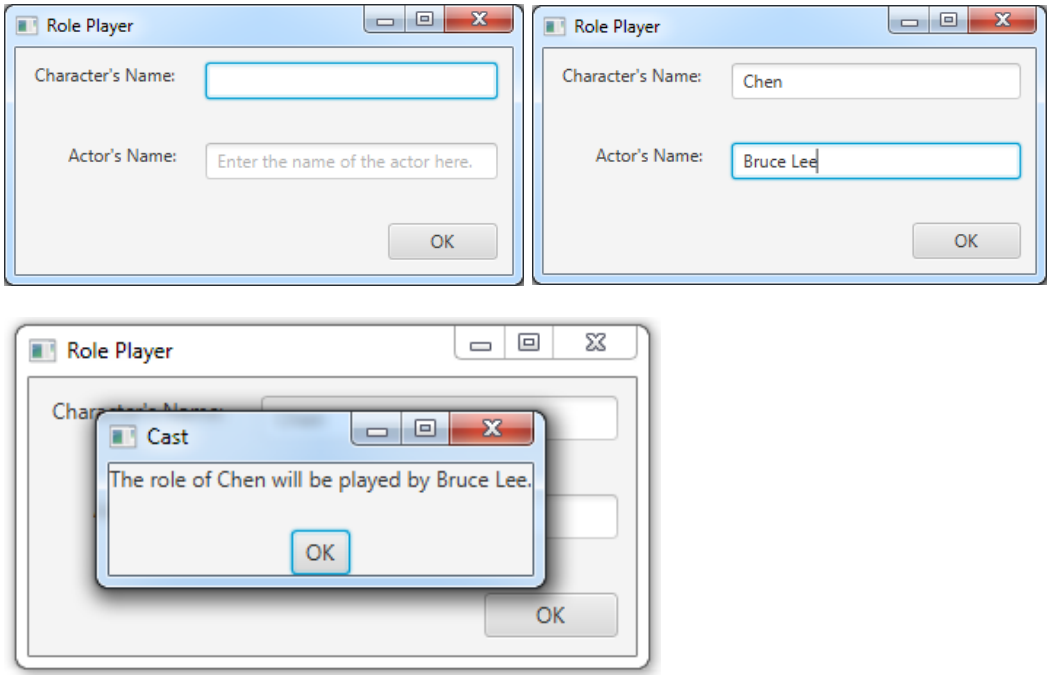
  public static void main(String[] args) {  launch(args);  }

  TextField txtCharacter;   TextField txtActor;
  @Override
  public void start(Stage primaryStage) {
    // Create the Character
    Label lblCharacter = new Label("Character's Name:");
    lblCharacter.setMinWidth(100);
    lblCharacter.setAlignment(Pos.BOTTOM_RIGHT);
    // Create the Character text field
    txtCharacter = new TextField();
    txtCharacter.setMinWidth(200);
    txtCharacter.setMaxWidth(200);
    txtCharacter.setPromptText(
                    "Enter the name of the character here.");
```

```
    // Create the Actor label
    Label lblActor = new Label("Actor's Name:");
    lblActor.setMinWidth(100);
    lblActor.setAlignment(Pos.BOTTOM_RIGHT);
    // Create the Actor text field
    txtActor = new TextField();
    txtActor.setMinWidth(200);
    txtActor.setMaxWidth(200);
    txtActor.setPromptText(
                    "Enter the name of the actor here.");
    // Create the OK button
    Button btnOK = new Button("OK");
    btnOK.setMinWidth(75);
    btnOK.setOnAction(e -> btnOK_Click());
    // Create the Character pane
    HBox paneCharacter = new HBox(20, lblCharacter,
                                              txtCharacter);
    paneCharacter.setPadding(new Insets(10));
    // Create the Actor pane
    HBox paneActor = new HBox(20, lblActor, txtActor);
    paneActor.setPadding(new Insets(10));
    // Create the Button pane
    HBox paneButton = new HBox(20, btnOK);
    paneButton.setPadding(new Insets(10));
    paneButton.setAlignment(Pos.BOTTOM_RIGHT);
    // Add the Character, Actor, and Button panes to a VBox
    VBox pane = new VBox(10, paneCharacter, paneActor,
                                                  paneButton);
    // Set the stage
    Scene scene = new Scene(pane);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Role Player");
    primaryStage.show();
  }
  public void btnOK_Click() {
    String errorMessage = "";
    if (txtCharacter.getText().length() == 0)
         errorMessage += "\nCharacter is a required field.";
    if (txtActor.getText().length() == 0)
         errorMessage += "\nActor is a required field.";
    if (errorMessage.length() == 0) {
      String message = "The role of " +
                        txtCharacter.getText() +
                       " will be played by " +
                       txtActor.getText() + ".";
      MessageBox.show(message, "Cast");
    }
```

```
    else {
        MessageBox.show(errorMessage, "Missing Data");
    }
  }
}
```

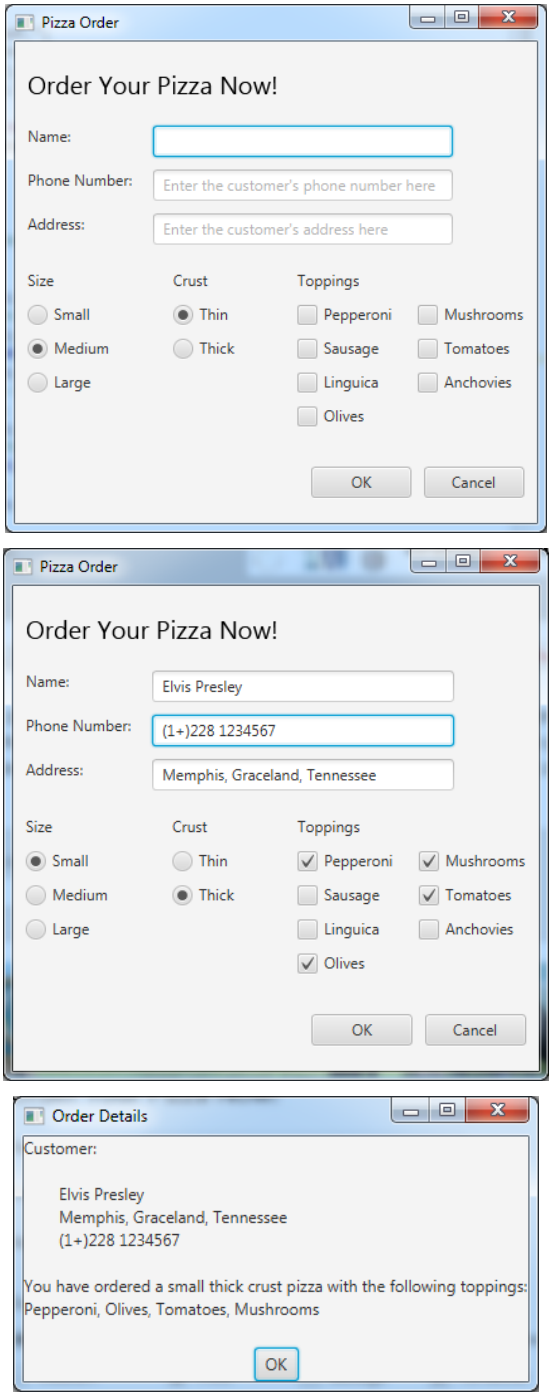
Slika 5.72. Kod klase za unos teksta i korišćenje klase MessageBox



Slika 5.73. Prikaz aplikacije za unos teksta i korišćenje klase MessageBox

5.3.3.6. Klase RadioButton i CheckBox

Klase `TextField`, `RadioButton` i `CheckBox` omogućuju: unos teksta (prethodni primer), odabiranje jedne iz skupa opcija te odabiranje bilo koje iz skupa ponuđenih opcija, respektivno. Neka je zahtev aplikacija za naručivanje pice čiji je izgled prikazan na slici 5.74. Razmislite kako biste rešili da se narudžba šalje na server TCP komunikacijom i da server potvrdi da je narudžba prihvaćena ili ne. Porudžbina sadrži: naslov, polja za unos: imena, broja telefona i adrese, biranje jedne od opcija veličine pice (mala, srednja, velika), biranje jedne od ponuđenih opcija debljine kore (debela, tanka), biranje bilo koje kombinacije preliva (feferoni, sos, kulen, masline, pečurke, paradajz, inćuni), programsku dugmad potvrde i odustajanja. Kada se odabere polje potvrde prikazuje se šta je sve odabrano (šta bi se slalo na server).



Slika 5.74. Prikaz aplikacije za naručivanje pice

Pripadni kod za naručivanje pice dat je na slici 5.75.

```
import javafx.application.*;    import javafx.stage.*;
import javafx.scene.*;          import javafx.scene.layout.*;
```

```
import javafx.scene.control.*; import javafx.geometry.*;
import javafx.scene.text.*;
public class PizzaOrder extends Application {
  public static void main(String[] args) { launch(args); }
  Stage stage;
  // Customer name, phone, and address fields
  TextField txtName;
  TextField txtPhone;
  TextField txtAddress;
  // Size radio buttons
  RadioButton rdoSmall;
  RadioButton rdoMedium;
  RadioButton rdoLarge;
  // Crust style radio buttons
  RadioButton rdoThin;
  RadioButton rdoThick;
  // Topping radio buttons
  CheckBox chkPepperoni;
  CheckBox chkSausage;
  CheckBox chkLinguica;
  CheckBox chkOlives;
  CheckBox chkMushrooms;
  CheckBox chkTomatoes;
  CheckBox chkAnchovies;
  @Override
  public void start(Stage primaryStage) {
    stage = primaryStage;
    // ----- Create the top pane -----
    Text textHeading = new Text("Order Your Pizza Now!");
    textHeading.setFont(new Font(20));
    HBox paneTop = new HBox(textHeading);
    paneTop.setPadding(new Insets(20, 10, 20, 10));
    // ---------- Create the customer pane ----------
    // Create the name label and text field
    Label lblName = new Label("Name:");
    lblName.setPrefWidth(100);
    txtName = new TextField();
    txtName.setPrefColumnCount(20);
     txtName.setPromptText("Enter the customer's name here");
    txtName.setMaxWidth(Double.MAX_VALUE);
    HBox paneName = new HBox(lblName, txtName);
    // Create the phone number label and text field
    Label lblPhone = new Label("Phone Number:");
    lblPhone.setPrefWidth(100);
   txtPhone = new TextField();
    txtPhone.setPrefColumnCount(20);
```

```
    txtPhone.setPromptText("Enter the customer's phone number
here");
    HBox panePhone = new HBox(lblPhone, txtPhone);
    // Create the address label and text field
    Label lblAddress = new Label("Address:");
    lblAddress.setPrefWidth(100);
    txtAddress = new TextField();
    txtAddress.setPrefColumnCount(20);
    txtAddress.setPromptText(
                        "Enter the customer's address here");
    HBox paneAddress = new HBox(lblAddress, txtAddress);
    // Create the customer pane
    VBox paneCustomer = new VBox(10, paneName, panePhone,
                                                paneAddress);
    // ---------- Create the order pane ----------
    // Create the size pane
    Label lblSize = new Label("Size");
    rdoSmall = new RadioButton("Small");
    rdoMedium = new RadioButton("Medium");
    rdoLarge = new RadioButton("Large");
    rdoMedium.setSelected(true);
    ToggleGroup groupSize = new ToggleGroup();
    rdoSmall.setToggleGroup(groupSize);
    rdoMedium.setToggleGroup(groupSize);
    rdoLarge.setToggleGroup(groupSize);
    VBox paneSize = new VBox(lblSize, rdoSmall, rdoMedium,
                                                  rdoLarge);
    paneSize.setSpacing(10);
    // Create the crust pane
    Label lblCrust = new Label("Crust");
    rdoThin = new RadioButton("Thin");
    rdoThick = new RadioButton("Thick");
    rdoThin.setSelected(true);
    ToggleGroup groupCrust = new ToggleGroup();
    rdoThin.setToggleGroup(groupCrust);
    rdoThick.setToggleGroup(groupCrust);
    VBox paneCrust = new VBox(lblCrust, rdoThin, rdoThick);
    paneCrust.setSpacing(10);
    // Create the toppings pane
    Label lblToppings = new Label("Toppings");
    chkPepperoni = new CheckBox("Pepperoni");
    chkSausage = new CheckBox("Sausage");
    chkLinguica = new CheckBox("Linguica");
    chkOlives = new CheckBox("Olives");
    chkMushrooms = new CheckBox("Mushrooms");
    chkTomatoes = new CheckBox("Tomatoes");
    chkAnchovies = new CheckBox("Anchovies");
```

```
    FlowPane paneToppings = new
      FlowPane(Orientation.VERTICAL,
          chkPepperoni, chkSausage, chkLinguica, chkOlives,
          chkMushrooms, chkTomatoes, chkAnchovies);
    paneToppings.setPadding(new Insets(10, 0, 10, 0));
    paneToppings.setHgap(20);
    paneToppings.setVgap(10);
    paneToppings.setPrefWrapLength(100);
    VBox paneTopping = new VBox(lblToppings, paneToppings);
    // Add the size, crust, and toppings pane
    // to the order pane
    HBox paneOrder = new HBox(50, paneSize, paneCrust,
                                              paneTopping);
    // Create the center pane
    VBox paneCenter = new VBox(20, paneCustomer, paneOrder);
    paneCenter.setPadding(new Insets(0, 10, 0, 10));
    // ---------- Create the bottom pane ----------
    Button btnOK = new Button("OK");
    btnOK.setPrefWidth(80);
    btnOK.setOnAction(e -> btnOK_Click());
    Button btnCancel = new Button("Cancel");
    btnCancel.setPrefWidth(80);
    btnCancel.setOnAction(e -> btnCancel_Click());
    Region spacer = new Region();
    HBox paneBottom = new HBox(10, spacer, btnOK,
                                            btnCancel);
    paneBottom.setHgrow(spacer, Priority.ALWAYS);
    paneBottom.setPadding(new Insets(20, 10, 20, 10));
    // ---------- Finish the scene ----------
    BorderPane paneMain = new BorderPane();
    paneMain.setTop(paneTop);
    paneMain.setCenter(paneCenter);
    paneMain.setBottom(paneBottom);
    // Create the scene and the stage
    Scene scene = new Scene(paneMain);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Pizza Order");
    primaryStage.show();
  }
  public void btnOK_Click() {
    // Create a message string with the customer information
    String msg = "Customer:\n\n";
    msg += "\t" + txtName.getText() + "\n";
    msg += "\t" + txtAddress.getText() + "\n";
    msg += "\t" + txtPhone.getText() + "\n\n";
    msg += "You have ordered a ";
    // Add the pizza size
```

```
    if (rdoSmall.isSelected()) msg += "small ";
    //getText() dalo bi Small
    if (rdoMedium.isSelected())msg += "medium ";
    if (rdoLarge.isSelected()) msg += "large ";
    if (rdoThin.isSelected()) msg +=
                            "thin crust pizza with ";  //kora
    if (rdoThick.isSelected())msg +=
                            "thick crust pizza with "; //kora
    String toppings = ""; //prelivi
    toppings = buildToppings(chkPepperoni, toppings);
    toppings = buildToppings(chkSausage, toppings);
    toppings = buildToppings(chkLinguica, toppings);
    toppings = buildToppings(chkOlives, toppings);
    toppings = buildToppings(chkTomatoes, toppings);
    toppings = buildToppings(chkMushrooms, toppings);
    toppings = buildToppings(chkAnchovies, toppings);
    if (toppings.equals("")) msg += "no toppings.";
    else                   msg += "the following toppings:\n"
                                + toppings;
    MessageBox.show(msg, "Order Details"); //od ranije
  }
  public String buildToppings(CheckBox chk, String msg) {
    if (chk.isSelected()) { if(!msg.equals(""))  msg+= ", ";
                            msg += chk.getText();
    }
    return msg;
  }
  public void btnCancel_Click() {stage.close();}
}
```

Slika 5.75. Kod za naručivanje pice

Klasa `PizzaOrder` proširuje klasu `Application`. Metoda `main` poziva metodu `launch(args)` gde su args argumenti komandne linije. Iza `main` metode navedene su deklaracije: pozornice, polja za unos, radio dugmadi i polja za potvrdu. Metoda `start` prihvata kao parametar referencu na pozornicu `primaryStage` koja se onda dodeljuje referenci `stage`.

Kreira se gornji deo porudžbine (okno-pane) `paneTop` klase `Hbox`, koga će činiti naslov `"Order Your Pizza Now!"`. Naslov je realizovan klasom `Text` (referenca `textHeading`), postavljanjem veličine fonta (`setFont(new Font(20))`) na 20 piksela. Granice `paneTop` elementa prema okolnim elementima su postavljene na 20,10,20,10 pozivom metode `setPadding` kojoj je prosleđen argument referenca na objekat `Insets` konstruisan sa navedenim vrednostima.

Deo porudžbine koji obuhvata prikaz labele (`Label`) za ime postavlja širinu labele na 100 pozivom metode labele `setPrefWidth(100)`. Polje za unos imena `txtName` klase `TextField` ima postavljenu preferiranu vrednost za broj

karaktera koji se prikazuju na 20 (`setPrefColumnCount(20)`). Prompt tekst predstavlja tekst koji će biti (podrazumevano svetlom sivom bojom) ispisan kada je `TextField` prazan, tako da služi kao uputstvo korisniku šta bi trebalo da se unese u dati `TextField` (`setPromptText("Enter the customer's name here")`) . Pri početku upisa u polje za tekst prompt tekst nestaje. Postavljena je i maksimalna širina za tekst polje `txtName` na takav način da se maksimalno prostire po širini kontejnera koji ga sadrži (`setMaxWidth(Double.MAX_VALUE)`). I labela `lblName` i tekst polje `txtName` dodaju se u `HBox panelName`.

Kao za labelu i tekst polje imena urađena su postavke za: labelu i tekst polje telefona te labelu i tekst polje adrese.

Sada su tri elementa `HBox` postavljeni u jedan `VBox`, `paneCustomer` sa inicijalnim razmakom među komponentama od 10 pikela.

Deo porudžbine koji se odnosi na biranje veličine pice sadrži labelu `lblSize` (`Label`) i tri radio dugmeta: `rdoSmall`, `rdoMedium`, `rdoLarge` (`RadioButton`) koji će biti smešteni u `VBox paneSize` sa međusobnim razmakom od 10 piskela. Sva tri navedena radio dugmeta postavljaju se u `ToggleGroup groupSize` metodom klase `RadioButton setToggleGroup(groupSize)` čime je omogućeno da ova tri dugmeta budu konstrolisana u smislu da samo jedno od navedenih može biti selektovano. Inicijalno je selektovano radio dugme za srednju veličinu pice pozivom metode `rdoMedium.setSelected(true).`

Kao i za prethodni deo koji se odnosi na odabiranje veličine urađen je i deo koji se odnosi na biranje debljine kore. Labela `lblCrust` i dva radio dugmeta `rdoThin` (selektovano), `rdoThick` kojima je postavljena kontrola `groupCrust` (`ToggleButton`) postavljeni su u `VBox paneCrust` (razmak među komponentama je 10 piskela).

Deo porudžbine koji se odnosi na biranje preliva sadrži labelu `lblToppings` te polja za potvdu (`CheckBox`): `chkPepperoni`, `chkSausage`, `chkLinguica`, `chkOlives`, `chkMushrooms`, `chkTomatoes`, `chkAnchovies`. Prelivi su postavljeni u `FlowPane paneToppings` sa vertikalnom orijentacijom (postavljanje ide po kolonama). Sada se labela `lblToppings` i `paneToppings` postavljaju u `VBox paneTopping`.

U `HBox paneOrder` se postavljaju delovi porudžbine za veličinu, debljinu kore i prelive (`paneSize`, `paneCrust`, `paneTopping`) sa međurazmakom od 50 piskela.

Sada se u `VBox paneCenter` postavljaju `paneCustomer` i `paneOrder` sa međurazmakom od 20 piksela. Postavlja se razmak za `paneCenter` od okolnih komponenti sa `setPadding(new Insets(0, 10, 0, 10)).`

Dva programska dugmeta `btnOK` i `btnCancel` sa predefinisanom širinom od 80 piksela imaju dodeljene, preko lambda izraza, metode koje će da obrađuju događaje klika na ovu programsku dugmad. Ovo je rešeno metodom

setOnAction. Za dugmad btnOK i btnCancel metode su btnOK_Click() i btnCancel_Click(), repsektivno.

Da bi se postiglo da postoji razmak sa leve strane stranice u delu u kome su programska dugmad btnOK i btnCancel korišćen je Region spacer. Region i navedena dva programska dugmeta smeštaju se u HBox paneBottom. Metodom HBox klase setHgrow(spacer, Priority.ALWAYS) postavlja se pravilo da se spacer uvek poveća (koliko može) po širini. Dodeljeni su i razmaci za paneBottom 20, 10, 20, 10, respektivno za gore, desno, dole, levo.

U BorderPane paneMain se smeštaju: paneTop, paneCenter, paneBottom metodama setTop, setCenter, setBottom, respektivno.

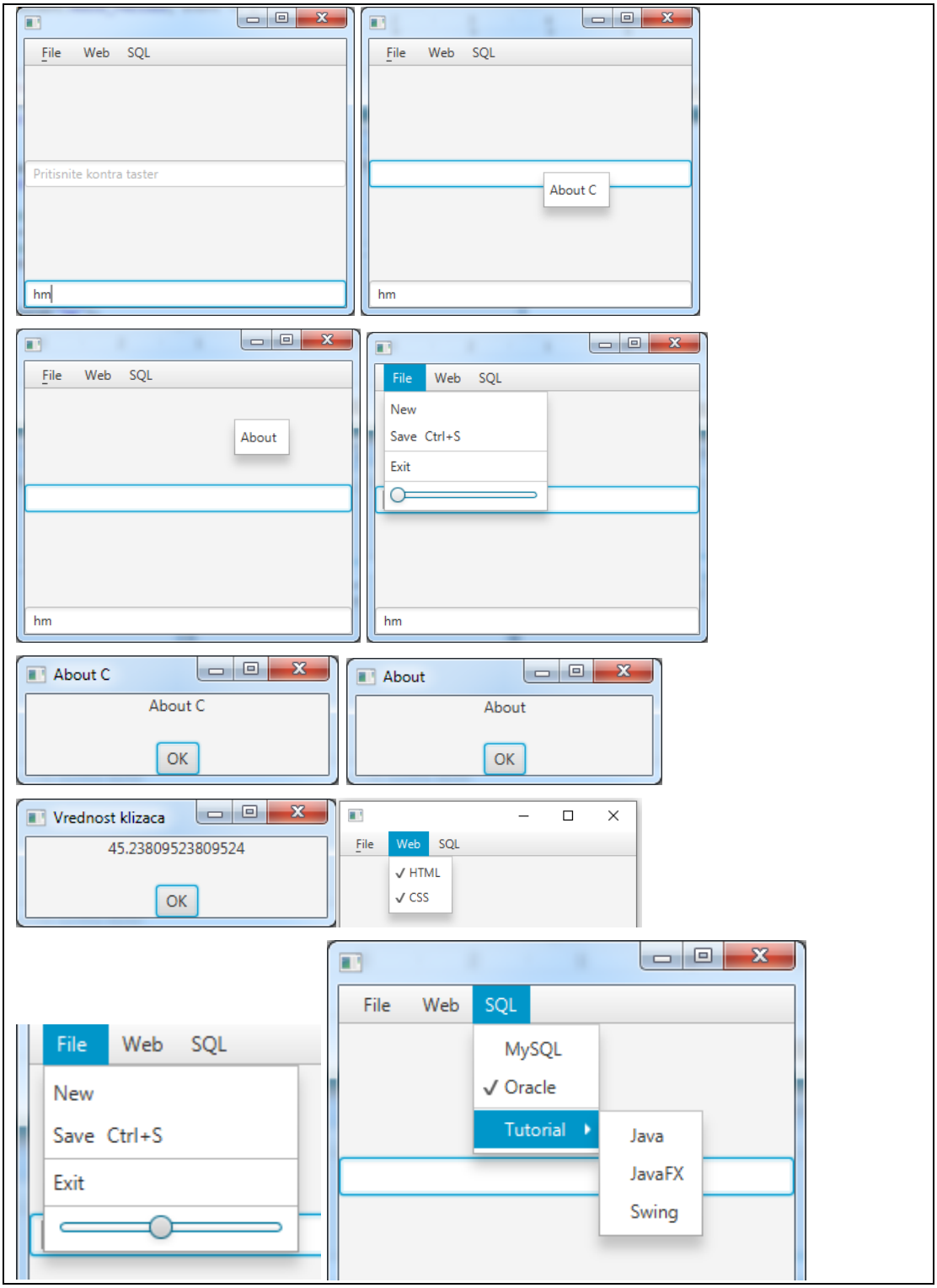
Na redu je postavljanje scene koja sadrži paneMain na pozornicu primaryStage kojoj je doljen naziv ("Pizza Order") i pozvana metoda za prikaz pozornice.

Metoda btnOK_Click(), koja obrađuje klik na programsko dugme btnOK, kreira String msg koji sadrži informacije unesenih podataka o imenu, adresi, telefonu, odabranoj veličini, odabranoj debljini kore i prelivima (ako nije odabran niti jedan preliv ispisuje i tu informaciju). Korišćene su metode: getText() za ime, adresu i telefon, isSelected() za radio dugmad i polja potvrde. Iskorišćena je metoda buildToppings kojoj se redom prosleđuju prelivi i gde se proverava da li je dati preliv selektovan te se u skladu sa tim vrši konatenacija odgovora (nabrajanja selektovanih preliva). Na kraju se poziva ranije opisana metoda show klase MessageBox da bi se prikazala porudžbina.

Metoda btnCancel_Click(), koja obrađuje klik na programsko dugme btnCancel, poziva metodu za zatvaranje pozornice (stage.close()).

5.3.3.7. JavaFX meniji

Koncept kreiranja menija u JavaFX tehnologiji sličan je rešenjima u pretohodnim tehnologijama. Izgled željene aplikacije dati je na slici 5.76. Kod za meni dat je na slici 5.77. Main metoda (na kraju koda) poziva preko metode launch, kojoj se prosleđuju argumenti iz komandne linije, metodu start koja prihvata kao argument referencu na Stage primaryStage. Kreira se BorderPane root koji se prosleđuje kao prvi parametar konstruktoru scene. Scena scene je veličine 300x250 piksela i bele je pozadine. Kreira se linija menija MenuBar menuBar, a onda se preferirana širina (svojstvo) vezuje za širinu pozornice primaryStage pozivom menuBar.prefWidthProperty().bind(primaryStage.widthProperty()). Linija menija se postavlja na gornji deo prozora (root.setTop(menuBar)). Kreira se meni (Menu) fileMenu gde je u konstruktoru prosleđen string "_File" koji će biti ispisan u traci menija. Donja crta ispred slova u stringu "_File" znači da će postojati prečica do menija fileMenu kao kombinacija tastera F+ctrl. Mnemoničko parsiranje je omogućeno pozivom metode fileMenu.setMnemonicParsing(true).



Slika 5.76. Izgled aplikacije sa menijima

Kreiraju se dve stavke menija (MenuItem) newMenuItem i saveMenuItem sa nazivima stavke menija "New" i "Save", respektivno. Za stavku saveMenuItem postavljen je akcelerator metodom setAccelerator kao kombinacija tastera S i tastera ctrl. Simoblička konstanta SHORTCUT_DOWN menja CONTROL_DOWN na Windows platformi i META_DOWN na Mac OS platformi. Za odabiranje stavke saveMenuItem biće pozvana metoda MessageBox.show navedena u lambda izrazu kao parametru metode setOnAction. Stavka menija exitMenuItem, čiji je naslov "Exit" reaguje na odabiranje tako da poziva metodu Platform.exit() čime se program nastavlja iza metode launch (u ovom primeru to je i izlaz iz aplikacije). Kreiraju se: klizač (Slider) slider i korisnička stavka menija (CustomMenuItem) kojoj je kao argument konstruktora prosleđen slider. Ako se želi da klikom na ovu korisnički stavku menija ne dođe do zatvaranja menija, onda se mora pozvati njena metoda setHideOnClick(false). Metoda koja će biti pozvana pri odabiranju ove stavke je MessageBox.show kojoj su prosleđeni: poruka koja pokazuje vrednost klizača i naslov ("Vrednost kizaca"). U fileMenu se dodaju stavke: newMenuItem, saveMenuItem, separator u vidu horizontalne crte (SeparatorMenuItem), exitMenuItem, separator i na kraju customMenuItem.

Meni webMenu, čiji je naslov "Web", sadrži dve stavke menija koje se selektuju (čekiraju) tipa CheckMenuItem: htmlMeniItem čiji je naslov "HTML" i cssMenuItem čiji je naslov "CSS". Obe stavke su selektovane pozivom metode setSelected(true). Dodavanje ovakve stavke u meni postiže se pozivom npr. webMenu.getItems().add(htmlMeniItem) za stavku htmlMeniItem.

Meni sqlMenu, čiji je naslov "SQL", sadrži dve stavke tipa RadioMenuItem: mysqlItem, oracleItem ("MySQL", "Oracle") koje se ponašaju kao radio dugmad tako da je za obe stavke postavljena ista kontrola tGroup (ToggleGroup) pri čemu je stavka oracleItem selektovana (setSelected(true)). Metodom setOnAction ostavljeno je da će obabiranjem stavke oracleItem biti pozvana metoda MessageBox.show("ORACLE","ORACLE"). U sqlMenu se dodaju: mysqlItem, oracleItem i separator pozivom sqlMenu.getItems().addAll(mysqlItem, oracleItem, new SeparatorMenuItem()). Kreira se meni tutorialSubMenu, čiji je naziv "Tutorial", koji će biti stavka (kaskadni meni) u okviru menija sqlMenu. U meni tutorialSubMenu dodaju se tri stavke tipa potvrde kao elementi tipa CheckMeniItem (naslovi: Java, JavaFX, Swing).

U traku menija dodaju se meniji: fileMenu, webMenu, sqlMenu.

Kreira se stavka menija itemAbout, čiji je naslov "About", i gde se njenim odabiranjem poziva metoda MessageBox.show("About", "About"). Ovo je relizovano metodm setOnAction gde je rešenje izvedeno preko anonimne klase (implementacijom EventHandler-a). Stavka itemAbout se prosleđuje kao parametar konstruktoru klase ContextMenu čime se kreira contextmenu.

Kontekst meni će se ponašati kao ranije rađen PopupMenu ako se za root (BorderPane) postavi rukovalac događajem pozivom metode root.addEventHandler koja prihvata dva argumenta: događaj (ContextMenuEvent.CONTEXT_MENU_REQUESTED) i lambda izraz koji poziva metodu show klase ContextMenu te sprečava dalju propagaciju događaja (event.consume()). Ova metoda show ima tri argumenta: prvi argument je kontejner u kome će se kontekst prikazati, drugi i treći argument određuju poziciju gde će se kontekst prikazati (event.getScreenX(),event.getScreenY()). Za skrivanje kontekst menija potrebno je postaviti rukovalac događajem pozivom metode root.addEventHandler , gde je prvi parametar događaj MouseEvent.MOUSE_PRESSED, a drugi parametar je lamda izraz gde se poziva metoda contextMenu.hide().

Slično kao itemAbout kreira se jedna stavka menija (MenuItem) itemAboutC čiji je naslov "About C". Odabiranjem ove stavke menija poziva se metoda MessageBox.show("About C", "About C"). Aktiviranje ove stavke menija biće vezano za tekst polje tf (TextField). Prompt tekst tekst polja tf je postavljen na "Pritisnite kontra taster". Konstruktoru kontekst menija contextMenuC prosleđuje se itemAboutC. Za tekst polje tf se veže kontekst meni pozivom (tf.setContextMenu(contextMenuC)). Tekst polje tf se dodaje u centar kontejnera root.

Jedno tekst polje tf2 se dodaje na dno kontejnera root. Postavlja se scena na pozornicu i pozornica se prikazuje.

```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.CheckMenuItem;
import javafx.scene.control.ContextMenu;
import javafx.scene.control.CustomMenuItem;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.RadioMenuItem;
import javafx.scene.control.SeparatorMenuItem;
import javafx.scene.control.Slider;
import javafx.scene.control.TextField;
import javafx.scene.control.ToggleGroup;
import javafx.scene.input.ContextMenuEvent;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyCodeCombination;
```

```
import javafx.scene.input.KeyCombination;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.BorderPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
public class Menu1 extends Application {
  @Override
  public void start(Stage primaryStage) {
    BorderPane root = new BorderPane();
    Scene scene = new Scene(root, 300, 250, Color.WHITE);
    MenuBar menuBar = new MenuBar();
    menuBar.prefWidthProperty().bind(
                              primaryStage.widthProperty());
    root.setTop(menuBar);
    // File menu - new, save, exit
    Menu fileMenu = new Menu("_File");
    fileMenu.setMnemonicParsing(true);
    MenuItem newMenuItem = new MenuItem("New");
    MenuItem saveMenuItem = new MenuItem("Save");
    saveMenuItem.setAccelerator(
       new KeyCodeCombination(KeyCode.S,
                           KeyCombination.SHORTCUT_DOWN));
       //CONTROL_DOWN or META_DOWN
    saveMenuItem.setOnAction(e->{
                              MessageBox.show("Snimanje",
                                              "Snimanje");});
    MenuItem exitMenuItem = new MenuItem("Exit");
    exitMenuItem.setOnAction(actionEvent-> Platform.exit());
    Slider slider = new Slider();
    CustomMenuItem customMenuItem = new
                                       CustomMenuItem(slider);
    customMenuItem.setHideOnClick(false);
    customMenuItem.setOnAction(e-> {
    MessageBox.show(""+slider.getValue(),
                                   "Vrednost klizaca"); }  );
    fileMenu.getItems().addAll(newMenuItem, saveMenuItem,
        new SeparatorMenuItem(), exitMenuItem,
        new SeparatorMenuItem(),
        customMenuItem);
    Menu webMenu = new Menu("Web");
    CheckMenuItem htmlMenuItem = new CheckMenuItem("HTML");
    htmlMenuItem.setSelected(true);
    webMenu.getItems().add(htmlMenuItem);
    CheckMenuItem cssMenuItem = new CheckMenuItem("CSS");
    cssMenuItem.setSelected(true);
    webMenu.getItems().add(cssMenuItem);
    Menu sqlMenu = new Menu("SQL");
```

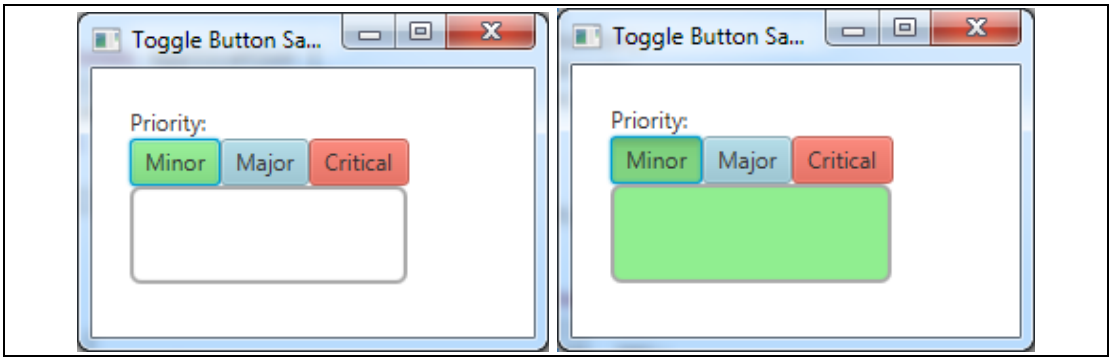
```
ToggleGroup tGroup = new ToggleGroup();
RadioMenuItem mysqlItem = new RadioMenuItem("MySQL");
mysqlItem.setToggleGroup(tGroup);
RadioMenuItem oracleItem = new RadioMenuItem("Oracle");
oracleItem.setToggleGroup(tGroup);
oracleItem.setSelected(true);
oracleItem.setOnAction(e->
                {MessageBox.show("ORACLE","ORACLE");});
sqlMenu.getItems().addAll(
                     mysqlItem, oracleItem,
                     new SeparatorMenuItem());
Menu tutorialManeu = new Menu("Tutorial");
tutorialManeu.getItems().addAll(
    new CheckMenuItem("Java"),
    new CheckMenuItem("JavaFX"),
    new CheckMenuItem("Swing"));
sqlMenu.getItems().add(tutorialManeu);
menuBar.getMenus().addAll(fileMenu, webMenu, sqlMenu);
MenuItem itemAbout = new MenuItem("About");
itemAbout.setOnAction(new EventHandler<ActionEvent>() {
  public void handle(ActionEvent e) {
    MessageBox.show("About", "About");
  }
});
ContextMenu contextMenu = new ContextMenu(itemAbout);
root.addEventHandler(
    ContextMenuEvent.CONTEXT_MENU_REQUESTED, event -> {
    contextMenu.show(root, event.getScreenX(),
                           event.getScreenY());
    event.consume();
});
root.addEventHandler(MouseEvent.MOUSE_PRESSED,
    event -> {
    contextMenu.hide();
});
MenuItem itemAboutC = new MenuItem("About C");
itemAboutC.setOnAction(new EventHandler<ActionEvent>() {
  public void handle(ActionEvent e) {
    MessageBox.show("About C", "About C");
  }
});
TextField tf = new TextField();
tf.setPromptText("Pritisnite kontra taster");
ContextMenu contextMenuC = new ContextMenu(itemAboutC);
//contextMenuC.show(tf, Side.BOTTOM, dx, dy);
//moze u f-iji
tf.setContextMenu(contextMenuC);
```

```
    root.setCenter(tf);
    TextField tf2 = new TextField("hm");
    root.setBottom(tf2);
    primaryStage.setScene(scene);
    primaryStage.show();
  }
  public static void main(String[] args) {  launch(args);  }
}
```

Slika 5.77. Listing aplikacije za menije

5.3.3.8. Klasa ToggleButton

Nekada je podesno koristiti posebno programsko dugme koja ima dva stanja: selektovano i neselektovano čime služi kao preklopnik. Na slici 5.78. prikazana su tri programska dugmeta tipa `ToggleButton` (`Minor`, `Major`, `Critical`). Ova tri dugmeta (preklopnika) imaju zajedničku kontrolu `ToggleGroup`, kao i radio dugmad. Klikom na `ToggleButton` isti menja prethodno stanje selekcije i utiče na pozadinsku boju pravougaonika. Kada su sva tri programska dugmeta tipa `ToggleButton` isključena, pozadina pravougaonika je bela. Poslednje koje se uključuje određuje boju pozadine pravougaonika.



Slika 5.78. Primene ToggleButton klase, slika levo: sva dugmad su isključena, slika desno: uključeno je dugme Minor.

Klasa `ToggleButtonSample` ima atribue: `rect` tipa `Rectangle` (pravougaonik) dimenzija 145x50 piksela, i labelu `label` čiji je naslov `"Priority:"`. Metoda `main` poziva `launch(args)`, a ona metodu `start` koja prihvata kao argument pozornicu `stage`. Kreira se scena kojoj se kao argument konstruktora prosleđuje referenca na objekt tipa `Group`. Postavlja se naslov pozornice i njene dimenzije (širina 250 piksela, visina 180 piksela). Pravougaoniku `rect` postavlja se bela boja pozadine (`setFill(Color.WHITE)`) te siva boja ograde (`setStroke(Color.DARKGRAY)`) debljine 2 piksela (`setStrokeWidth(2)`).

Kreira se `group` tipa `ToggleGroup` i njegovom svojstvu "preklapanja" postavlja osluškivač pozivom `group.selectedToggleProperty().addListener`.

Argument metode addListener je anonimna klasa koja implementira ChangeListener<Toggle> tako što definiše metodu changed koja prihvata tri argumenta: ObservableValue<? Extends Toggle> (vidi poglavlje generici) ov, Toggle toggle, Toggle new_toggle. Ako je nova situacija takva da ništa nije selektovano (new_toggle==null), onda se pravougaonik rect popunjava belom bojom (setFill(Color.WHITE)), inače će se pravougaonik popuniti bojom koja se dohvata tako da se za group poziva metoda getSelectedToggle(), a onda se od tog selektovanog elementa uzimaju postavljeni korisnički podaci (getUserData()). Korsnički podaci se za svaki ToggleButton postavljaju pozivom metode setUserData npr. setUserData(Color.LIGHTBLUE).

Kreiraju se tri elementa tipa ToggleButton tb1, tb2 i tb3. Svakom se dodeljuje ista kontrola (setToggleGroup(group)), postavljaju korisniči podaci o boji, postavlja boja pozadine (npr. setStyle("-fx-base: lightgreen;")) i eventualna početna selekcija. Elementi tb1, tb2, tb3 se dodaju u HBox hbox. Pravougaoniku rect se postavljaju zaobljeni krajevi (setArcHeight(10), setArcWidth(10)). Labela label, hbox i rect se dodaju u VBox vbox sa ogradom Insets(20,10,10,20). Uzima se korenski element scene kastovanjem poziva (Group) scene.getRoot(), a onda njegovi elementi (getChildren()) i dodaje se vbox. Scena se postavlja na pozornicu i poziva prikaz pozornice. Pripadni kod dat je na slici 5.79.

```
import javafx.application.Application;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.Toggle;
import javafx.scene.control.ToggleButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;

public class ToggleButtonSample extends Application {
   Rectangle rect = new Rectangle(145, 50);
   private static final Label label =
                                 new Label ("Priority:");

   public static void main(String[] args) { launch(args); }
   @Override
   public void start(Stage stage) {
```

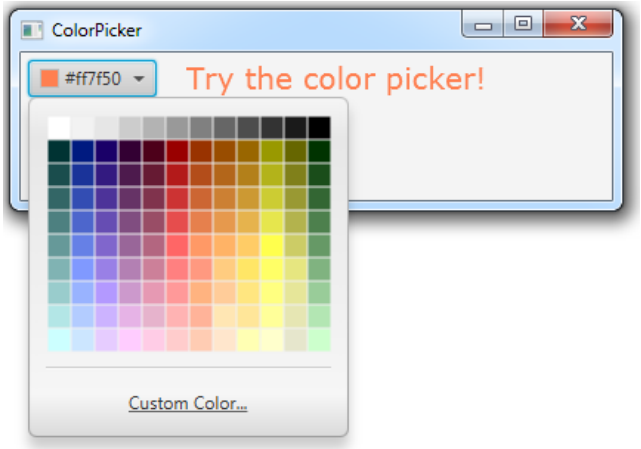
```
        Scene scene = new Scene(new Group());
        stage.setTitle("Toggle Button Sample");
        stage.setWidth(250);          stage.setHeight(180);
        rect.setFill(Color.WHITE);
        rect.setStroke(Color.DARKGRAY);
        rect.setStrokeWidth(2);
        final ToggleGroup group = new ToggleGroup();
        group.selectedToggleProperty().addListener(
          new ChangeListener<Toggle>(){
            public void changed(
              ObservableValue<? extends Toggle> ov,
              Toggle toggle, Toggle new_toggle) {
                if (new_toggle == null)
                   rect.setFill(Color.WHITE);
                else rect.setFill(
                  (Color)
                  group.getSelectedToggle().getUserData() );
            }
        });
        ToggleButton tb1 = new ToggleButton("Minor");
        tb1.setToggleGroup(group);
        tb1.setUserData(Color.LIGHTGREEN);
        tb1.setSelected(true);
        tb1.setStyle("-fx-base: lightgreen;");
        ToggleButton tb2 = new ToggleButton("Major");
        tb2.setToggleGroup(group);
        tb2.setUserData(Color.LIGHTBLUE);
        tb2.setStyle("-fx-base: lightblue;");
        ToggleButton tb3 = new ToggleButton("Critical");
        tb3.setToggleGroup(group);
        tb3.setUserData(Color.SALMON);
        tb3.setStyle("-fx-base: salmon;");
        HBox hbox = new HBox();
        hbox.getChildren().add(tb1);
        hbox.getChildren().add(tb2);
        hbox.getChildren().add(tb3);
        rect.setArcHeight(10);
        rect.setArcWidth(10);
        VBox vbox = new VBox();
        vbox.getChildren().add(label);
        vbox.getChildren().add(hbox);
        vbox.getChildren().add(rect);
        vbox.setPadding(new Insets(20, 10, 10, 20));
        ((Group) scene.getRoot()).getChildren().add(vbox);
        stage.setScene(scene);
        stage.show();
    }
```

```
}
```

Slika 5.79. Listing aplikacije koja koristi ToggleButton

5.3.3.9. Klasa ColorPicker

Klasa `ColorPicker` omogućuje selekciju boje iz standardnog dijaloga za selekciju boje. Izgled aplikacije koja koristi `ColorPicker` data je na slici 5.80.



Slika 5.80. Izgled aplikacije koja koristi ColorPicker

U okviru klase `ColorPickerSample` u metodi `start`, koja prihvata referencu na pozornicu, postavlja se naslov pozornice (`"ColorPicker"`), kreira se scena koja ima kao korenski element ima neimenovani `HBox` i dimenzija je 400x100 piksela. Korenski element se može dohvatiti sa scene kastovanjem na `HBox` povratne vrednosti metode `getRoot()` i dodeliti referenci `box`. Postavljaju se odstojanja za element `box` od ostalih elemenata (`Insets(5)`). Kreira se `colorPicker` tipa `ColorPicker` kome se nakon toga postavlja inicijalna boja (`setValue.(Color.CORAL)`). Referenca `colorPicker` je definisana kao `final`, tako da se ne može dodeliti da referencira drugi objekat. Kreira se i text tipa `Text` (predstavlja geometrijski oblik) koji ispisuje `"Try the color picker!"` u fontu `Verdana` veličine 20 piksela (`setFont`) i obojeno bojom (`setFill`) koja se uzima od elementa `colorPicker` metodom `getValue()`. Događaj selekcije boje u elementu `colorPicker` obrađuje se tako da se `text` oboji odabranom bojom (`colorPicker.setOnAction(e->{text.setFill(colorPicker.getValue());}`)). Elementi `colorPicker` i `text` se dodaju u `box`, postavlja se scena na pozornicu i prikazuje pozornica. Videti sliku 5.81.

```
import javafx.application.Application; import javafx.event.*;
import javafx.scene.Scene; import
javafx.scene.control.ColorPicker;
import javafx.geometry.Insets; import javafx.scene.layout.HBox;
import javafx.scene.paint.Color; import javafx.scene.text.*;
```

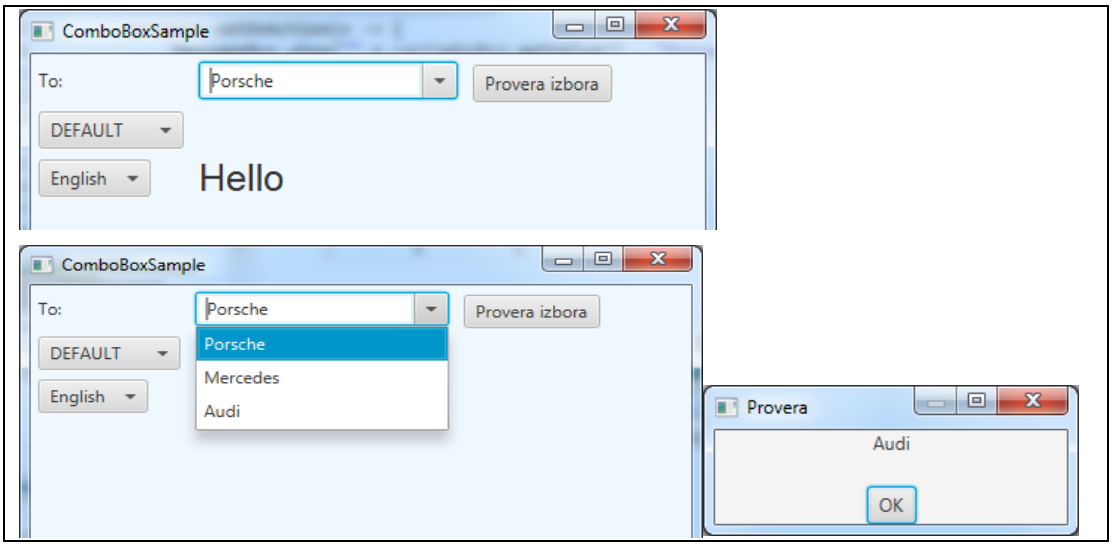
```
import javafx.stage.Stage;
public class ColorPickerSample extends Application {
  public static void main(String[] args) { launch(args); }
  @Override
  public void start(Stage stage) {
    stage.setTitle("ColorPicker");
    Scene scene = new Scene(new HBox(20),400,100);
    HBox box = (HBox)scene.getRoot();
    box.setPadding(new Insets(5, 5, 5, 5));

    final ColorPicker colorPicker = new ColorPicker();
    colorPicker.setValue(Color.CORAL);
    final Text text = new Text("Try the color picker!");
    text.setFont(Font.font ("Verdana", 20));
    text.setFill(colorPicker.getValue());
    colorPicker.setOnAction(
                e->{text.setFill(colorPicker.getValue());});
    box.getChildren().addAll(colorPicker, text);
    stage.setScene(scene);
    stage.show();
  }
}
```

Slika 5.81. Listing aplikacije koja koristi ColorPicker

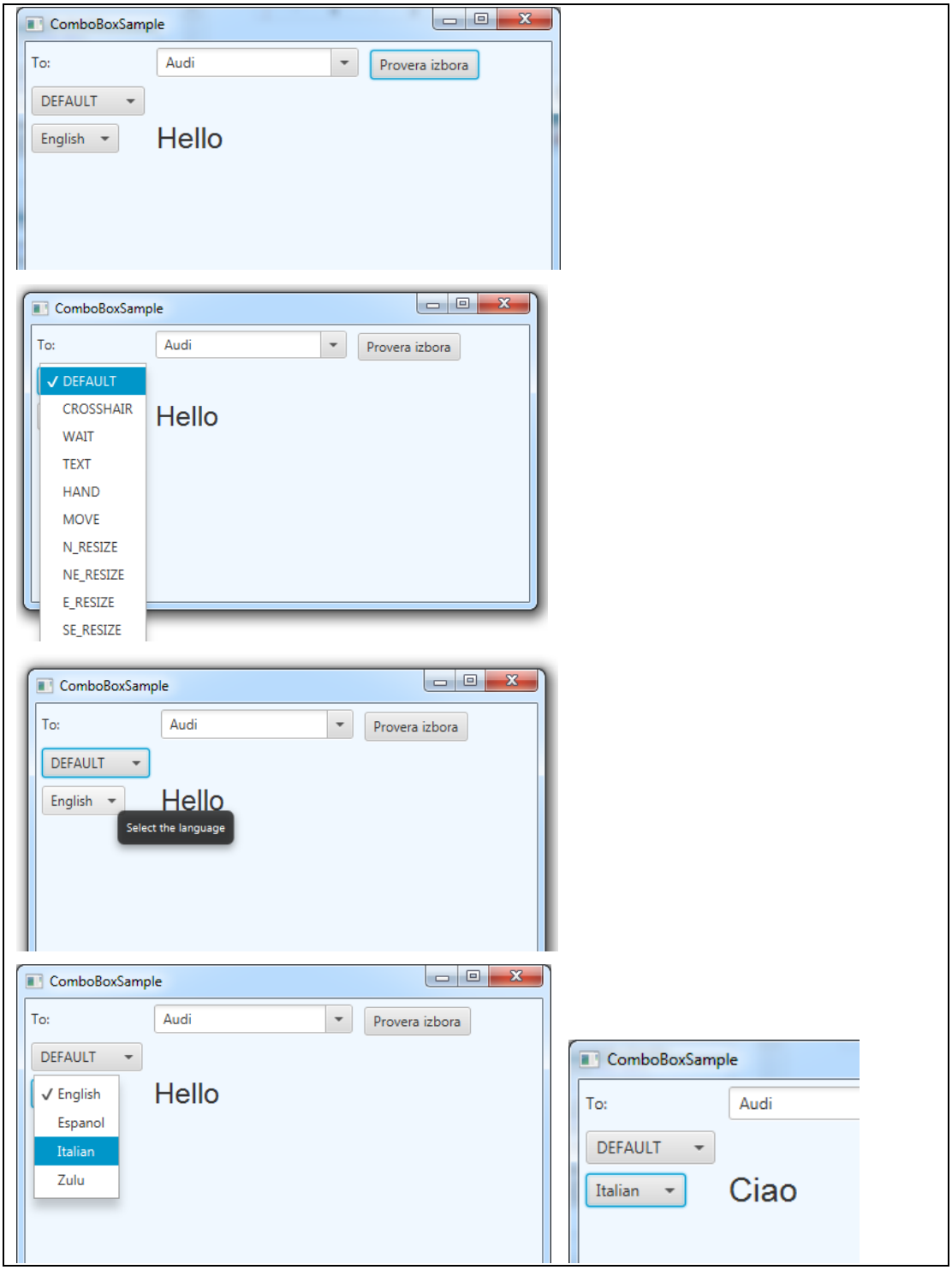
5.3.3.10. Klase ComboBox i ChoiceBox

Na slikama 5.82. i 5.83. dat je izgled aplikacije koja koristi klase `ComboBox` i `ChoiceBox`. Omogućeno je: biranje automobila i provera izabranog modela, biranje kursora te prikaz pozdrava na odabranom jeziku.



Slika 5.82. Izgled aplikacije koja koristi ComboBox, ChoiceBox (odabiranje

automobila)



Slike 5.83. Izgled aplikacije koja koristi ComboBox, ChoiceBox (biranje kursora, biranje jezika za prikaz pozdravne poruke)

Kod aplikacije (korisničke klase) Combo dat je na slici 5.84. Neka korisnička klasa Combo ima atribut cursors tipa ObservableList. Ova lista je popunjena pozivom metode FXCollections.observableArrayList čiji su parametri simboličke konstante kursora (Cursor.DEFAULT, ... , Cursor.NONE). Main metoda ima poziv launch(args), a metoda start prihvata referencu na pozornicu. U metodi start, postavlja se naslov pozornice i kreira scena sa neimenovanim Group elementom kao korenskim elementom. Veličina scene je 450x250 piksela, a boja pozadine Color.ALICEBLUE. Kreira se carComboBox tipa ComboBox kome se dodaju tri elementa ("Porsche", "Mercedes", "Audi"), postavlja se da je moguće editovanje i postavlja se inicijalna vrednost ("Porsche").

Svojstvu valueProperty elementa carComboBox dodaje se osluškivač koji implementira ChangeListener<String> koji je parametar poziva metode (carComboBox.valueProperty(). addListener). Metoda changed prihvata tri argumenta: ObservableValue ov, String t, String t1 i ispisuje u konzoli vrednosti ovih argumenata. Npr. ako je prethodno bio izabran Mercedes, a onda se odabere Audi ispis bi moga biti:
ObjectProperty [bean: ComboBox@31fe1e14[styleClass=combo-box-base combo-box], name: value, value: Audi]
Mercedes
Audi

Zadatak programskog dugmeta provera je da klikom na isto bude pozvana metoda MessageBox.show koja bi ispisala selektovanu vrednost elementa carComboBox. Argumenti ove show metode su: carComboBox.getValue() i „Provera").

Element choiceBox tipa ChoiceBox kreiran je pozivom konstruktora ChoiceBox<Cursor>(cursors) čime je choiceBox popunjen listom cursors. Pozivom metode choiceBox.setValue(Cursor.DEFAULT) postavlja se podrazumevana vrednost odabranog kursora.

Kreira se labela čiji je naslov „Hello" i font arial veličine 25 piksela. Niz stringova greetings inicijalno ima četiri stringa čije je značenje pozdrav na različitim jezicima (engleskom, španskom, italijanskom i zulu). Kreira se prevod tipa ChoiceBox koji sadrži listu stringova naziva navedenih jezika kreiranu pomoću FXCollections.observableArrayList. Inicijalno je postavljen engleski („English"). Tekst objašnjenja (tooltip) postavlja se metodom setTooltip kojoj se prosleđuje referenca na objekt tipa Tooltip. Parametar Tooltip konstruktora je string "Select the language". Poziva se metoda prevod.getSelectionModel(). selectedIndexProperty() kojom se uzima selekcioni model (po indeksu ili po članovima), a onda svojstvo za selektovani indeks kome se dodaje osluškivač (addListener). Ovaj oslučkivač implementira ChangeListener<Number> definišići metodu changed koja ima tri

parametra: `ObservableValue ov, Number value, Number new_value,` i koja postavlja tekst labele `label` na novu izabranu vrednost tako da uzme novi selektovani indeks (`new_value`) koji određuje string u nizu `greetings` (`greetings[new_value.intValue()]`).

U `GridPane grid` sa vertikalnim rastojanjem komponenata od 4 piskela, horizontalnim rastojanjem komponenata od 10 piskela te okolnim rastojanjem od 5 piskela se dodaju sledeći elementi:

- labela, čiji je naslov „To: " na kolonu 0, red 0;
- carComboBox na kolonu 1, red 0;
- provera na kolonu 2, red 0, proširenje na 2 kolone i proširenje na 2 reda;
- choiceBox na kolonu 0, red 2;
- prevod na kolonu 0, red 3;
- label na kolonu 1 red 1.

Uzima se korenski element scene, u koji se dodaje grid, scena se postavlja na pozornicu i prikazuje se pozornica. Svojstvo kursor scene se veže za član koji je izabran u `choiceBox` elementu. Ovo vezivanje je rešeno pozivom `choiceBox.getSelectionModel().selectedItemProperty()`).

```
import javafx.scene.shape.Rectangle;
import javafx.application.Application;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.Cursor;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.ComboBox;
import javafx.scene.control.ContentDisplay;
import javafx.scene.control.Label;
import javafx.scene.control.ListCell;
import javafx.scene.control.ListView;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.util.Callback;
import javafx.scene.control.Tooltip;
public class Combo extends Application {
  ObservableList cursors =
```

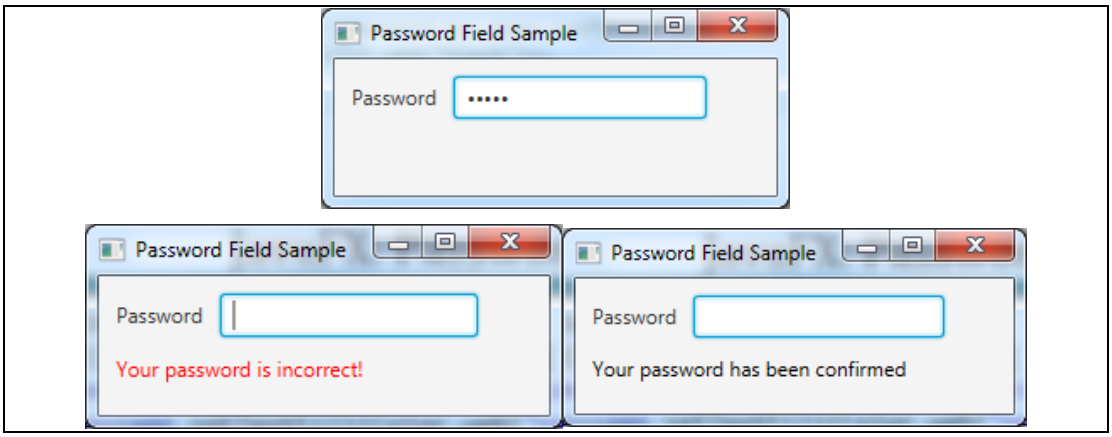
```
FXCollections.observableArrayList(
  Cursor.DEFAULT, Cursor.CROSSHAIR, Cursor.WAIT,
  Cursor.TEXT, Cursor.HAND, Cursor.MOVE, Cursor.N_RESIZE,
  Cursor.NE_RESIZE, Cursor.E_RESIZE, Cursor.SE_RESIZE,
  Cursor.S_RESIZE, Cursor.SW_RESIZE, Cursor.W_RESIZE,
  Cursor.NW_RESIZE, Cursor.NONE);
public static void main(String[] args)
                              {Application.launch(args); }
@Override
public void start(Stage stage) {
  stage.setTitle("ComboBoxSample");
  Scene scene = new Scene(new Group(), 450, 250);
  scene.setFill(Color.ALICEBLUE);
  ComboBox carComboBox = new ComboBox();
  carComboBox.getItems().addAll("Porsche", "Mercedes",
                                                "Audi");
  carComboBox.setEditable(true);
  carComboBox.setValue("Porsche");
  //za null nema podrazumevanog. aut.
  carComboBox.valueProperty().addListener(
     new ChangeListener<String>() {
        @Override
        public void changed(ObservableValue ov,
                              String t, String t1) {
          System.out.println(ov); System.out.println(t);
          System.out.println(t1);
        }
     }
  );
  Button provera = new Button("Provera izbora");
  provera.setOnAction(e -> {
    MessageBox.show(""+carComboBox.getValue(), "Provera");
  });
  ChoiceBox choiceBox = new ChoiceBox<Cursor>(cursors);
  choiceBox.setValue(Cursor.DEFAULT);
  Label label = new Label("Hello");
  label.setStyle("-fx-font: 25 arial;");
  final String[] greetings = new String[] {
                    "Hello", "Hola", "Ciao", "Sawubona" };
  final ChoiceBox prevod = new ChoiceBox
    (FXCollections.observableArrayList(
                "English", "Espanol", "Italian", "Zulu"));
  prevod.setValue("English");
  prevod.setTooltip(new Tooltip("Select the language"));
  prevod.getSelectionModel().selectedIndexProperty().
    addListener(
      new ChangeListener<Number>() {
```

```
            public void changed(
              ObservableValue ov,
              Number value, Number new_value) {
               label.setText(greetings[new_value.intValue()]);
            }
        }
    );
    GridPane grid = new GridPane();
    grid.setVgap(4);    grid.setHgap(10);
    grid.setPadding(new Insets(5, 5, 5, 5));
    grid.add(new Label("To: "), 0, 0);
    grid.add(carComboBox, 1, 0);
    grid.add(provera , 2, 0, 2, 2);
    grid.add(choiceBox, 0, 2);
    grid.add(prevod, 0, 3);
    grid.add(label, 1, 3);
    Group root = (Group) scene.getRoot();
    root.getChildren().add(grid);
    stage.setScene(scene);
    stage.show();
    scene.cursorProperty().bind(
      choiceBox.getSelectionModel().selectedItemProperty());
  }
}
```

Slika 5.84. Listing Combo aplikacije

5.3.3.11. Klasa PasswordField

Klasa `PasswordField` se koristi kao polje za unos lozinke gde se uneseni karakteri prikazuju kao kružići. Na slici 5.85. dat je scenario gde se unosi lozinka, a onda se u labeli smeštenoj ispod polja za unos lozinke ispisuje tekst potvrde ispavnosti unete lozinke. Poruka da je lozinka nekorektna se ispisuje crvenim tekstom, dok se poruka da je lozinka potvrđena ispisuje crnim teksom.



Slika 5.85. Izgled aplikacije za unos lozinke

Kod aplikacije Password dat je na slici 5.86. Klasa Password, kao i u ranije datim primerima, u main metodi poziva launch(args), a u start metodi prihvata referencu na pozornicu. Ova klasa ima atribut message tipa Label. U metodi start kreirana je scena dimenzija 260x80 piksela. Korenski element scene je root tipa Group. Postavlja se scena na pozornicu i postavlja naslov pozornice ("Password Field Sample"). Kreira se VBox vb sa postavljenim granicama prema okolnim elementima (setPadding(Insets(10,0,0,10)) i postavljenim razmakom među elementima u vb od 10 piksela. U HBox hb postavljeno je da je razmak među komponentama u hb 10 piksela i da su komponente centrirane i levo (setAlignment(Pos.CENTER_LEFT)). Labela label, čiji je naslov „Password" i PasswordField pb biće postavljeni u hb, a onda hb i message u vb koji se postavlja kao korenski element scene (scene.setRoot(vb)). Za pb je postavljena akcija (setOnAction) u kojoj se ispituje uneseni tekst lozinke, a onda prema korektnosti se u labeli message ispisuje crvenim tekstom da je lozinka nekorektna ili crnim tekstom da je lozinka potvrđena. Na kraju metode start se prikazuje pozornica. Videti sliku 5.86.

```
import javafx.application.Application;
import javafx.geometry.Insets; import javafx.geometry.Pos;
import javafx.scene.Group; import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.layout.HBox; import javafx.scene.layout.VBox;
import javafx.scene.paint.Color; import javafx.stage.Stage;
public class Password extends Application {
  final Label message = new Label("");
  @Override
  public void start(Stage stage) {
    Group root = new Group();
    Scene scene = new Scene(root, 260, 80);
    stage.setScene(scene);
    stage.setTitle("Password Field Sample");
    VBox vb = new VBox();
    vb.setPadding(new Insets(10, 0, 0, 10));
    vb.setSpacing(10);
    HBox hb = new HBox();
    hb.setSpacing(10);
    hb.setAlignment(Pos.CENTER_LEFT);
    Label label = new Label("Password");

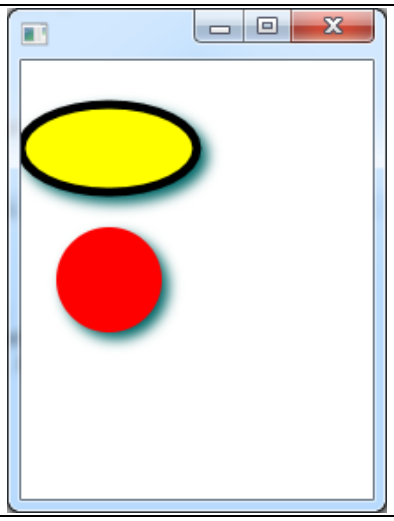
    final PasswordField pb = new PasswordField();
     pb.setOnAction(e->{
        if (!pb.getText().equals("abc")) {
          message.setText("Your password is incorrect!");
          message.setTextFill(Color.web("red"));
```

```
        } else {
          message.setText(
                    "Your password has been confirmed");
          message.setTextFill(Color.web("black"));
        }
        pb.setText("");
    });
    hb.getChildren().addAll(label, pb);
    vb.getChildren().addAll(hb, message);
    scene.setRoot(vb);
    stage.show();
  }
  public static void main(String[] args) { launch(args);   }
}
```

Slika 5.86. Listing aplikacije Password

5.3.3.12. Crtanje u JavaFX.

JavaFX omogućuje i crtanje sa raznim svojstvima. Na slici 5.87. dat je izgled nacrtanih: elipse i kruga sa svojstvima popunjavanja, debljinom crtanja, pozicioniranja, senkama te postavljanjem keširanja kojim se postižu veće brzine renderovanja.



Slika 5.87. Crtanje u JavaFX.

Na slici 5.88. dat je listing aplikacije `CirclePrimer`. U klasi `CirclePrimer` u `main` metodi se poziva `launch(args)`, a u `start` metodi prihvata se referenca na pozornicu. Scena je veličine 200x250 piksela i bele pozadine čiji je korenski element `root` tipa `Group`.

Kreira se referenca `ds` na objekat klase `DropShadow`. Ova klasa omogućuje rad sa senkama tako da se postavljaju pomeraji senke po `x` i po `y` osi te boja senke metodama `setOffsetX`, `setOffsetY` i `setColor`, respektivno. U datom primeru

pomeraj za x i za z je 4 piksela, a boja cijan (0.0, 0.4, 0.4).

Kreira se krug c tipa `Circle` kome se postavljaju: efekti senke (`setEffect(ds)`), x koordinata centra kruga (`setCenterX(50.0)`), y koordinata centra kruga (`setCenterY(125.0)`), radijus (`setRadius(30.0)`), popunjavanje crvenom bojom (`setFill(Color.RED)`) te keširanje zbog povećanja brtzine renderovanja (`setCache(true)`).

Kreira se elipsa e tipa `Ellipse` kojoj se postavljaju: x koordinata centra elipse (`setCenterX(50.0)`), y koordinata centra elipse (`setCenterY(50.0)`), x radijus (`setRadiusX(50.0)`), y radius (`setRadiusY(25.0)`), popunjavanje žutom bojom (`setFill(Color.RED)`), crna boja crtanja (`setStroke(Color.BLACK)`), debljina pera kojim se crta (`setStrokeWidth(5)`) te senka `ds` (`setEffect(ds)`).

U `root` se dodaju krug i elipsa, postavlja scena na pozornicu i prikazuje pozornica.

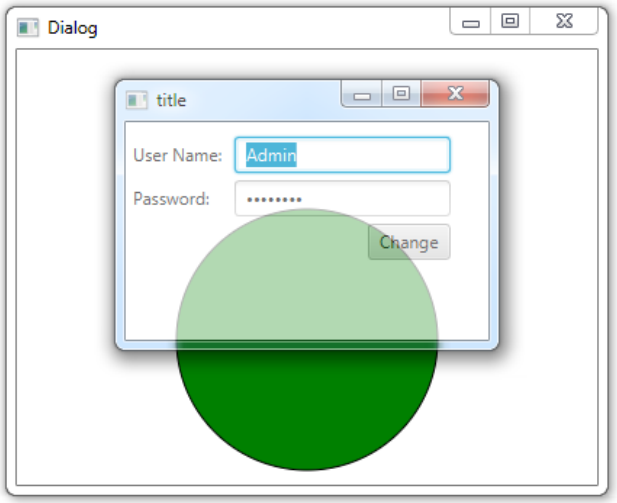
```
import javafx.application.Application; import javafx.scene.Group;
import javafx.scene.Scene; import javafx.scene.effect.DropShadow;
import javafx.scene.paint.Color; import javafx.scene.shape.Circle;
import javafx.scene.shape.Ellipse; import javafx.stage.Stage;
public class CirclePrimer extends Application {
  public static void main(String[] args) { launch(args); }
  public void start(Stage primaryStage) {
    Group root = new Group();
    Scene scene = new Scene(root, 200, 250, Color.WHITE);
    DropShadow ds = new DropShadow();
    ds.setOffsetX(4.0);    ds.setOffsetY(4.0);
    ds.setColor(Color.color(0.0, 0.4, 0.4)); //cyan
    Circle c = new Circle();
    c.setEffect(ds);
    c.setCenterX(50.0);
    c.setCenterY(125.0);
    c.setRadius(30.0);
    c.setFill(Color.RED);
    c.setCache(true);
    Ellipse e = new Ellipse();
    e.setCenterX(50.0f);
    e.setCenterY(50.0f);
    e.setRadiusX(50.0f);
    e.setRadiusY(25.0f);
    e.setFill(Color.YELLOW);
    e.setStroke(Color.BLACK);
    e.setStrokeWidth(5);
    e.setEffect(ds);
    root.getChildren().addAll(c, e);
```

```
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

Slika 5.88. Listing aplikacije CirclePrimer

5.3.3.13. Poziv dijaloga

Ideja je da aplikacija prikaže i drugu pozornicu čija će se veličina prilagoditi sceni i koji će imati svojstvo providnosti. Vidi sliku 5.89.



Slika 5.89. Aplikacija koja poziva dijalog (prikazuje i drugu pozornicu)

Klasa `CallDialog` proširuje klasu `Application`, u `main` metodi poziva `launch(args)`, a u `start` metodi prihvata se referenca na pozornicu `primaryStage`. Scena je dimenzija 400x300 piksela, bele pozadine i ima korenski element `root` tipa `Group`. Na koordinatama 200,200 je zeleni krug crtan crnim perom radijusa 90. U `root` se dodaje krug, postavlja se scena na pozornicu i prikazuje pozornica. Nakon ovoga kreira se nova pozornica `myDialog` koja je referenca na objekat tipa `MyDialog` čijem konstruktoru se prosleđuje pozornica `primaryStage`. Pozornica `myDialog` se podešava prema sceni (`sizeToScene()`) i prikazuje. Videti sliku 5.90.

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
public class CallDialog extends Application {
    public static void main(String[] args) { launch(args); }
    public void start(final Stage primaryStage) {
```

```
        primaryStage.setTitle("Dialog");
        Group root = new Group();
        Scene scene = new Scene(root, 400, 300,Color.WHITE);
        Circle c = new Circle();
        c.setCenterX(200);
        c.setCenterY(200);
        c.setRadius(90);
        c.setFill(Color.GREEN);
        c.setStroke(Color.BLACK);

        root.getChildren().add(c);
        primaryStage.setScene(scene);
        primaryStage.show();

        Stage myDialog = new MyDialog(primaryStage);
        myDialog.sizeToScene();
        myDialog.show();
    }
}
```

Slika 5.90. Klasa koja zove dijalog

Klasa `MyDialog` proširuje klasu `Stage`. Konstruktor klase `MyDialog` prihvata referencu na pozornicu `owner`. U ovom konstruktoru poziva se konstruktor superklase `Stage`, a onda metodom `initOwner(owner)` postavlja kojoj pozornici ("prozoru") pripada `MyDialog`. Postavljaju se: naziv pozornice (`setTitle("title")`) i neprovidnost pozornice (`setOpacity(.80)`). Kreira se scena `scene` veličine 250x150 piksela, bele pozadine i čiji je korenski element `root` tipa `Group`. Postavlja se scena na pozornicu (`setScene(scene)`). Kreira se gridpane tipa `GridPane` čija su odstojanja od drugih komponenti 5 piksela (`setPadding(Insets(5))`, a razmak između komponenti u kontejneru `gridpane` 5 piksela horizontalno i 5 piksela vetikalno (`setHgap(5)`, `setVgap(5)`).

U `gridpane` se dodaju:

- labela `userNameLbl` čiji je naslov `"User Name: "` u kolonu 0 i red 1;
- labela `passwordLbl` čiji je naslov `"Password: "` u kolonu 0, red 2;
- tekst polje `usernameFld` čiji je naslov `"Admin"` u kolonu 1, red 1;
- polje za lozinku `passwordFld` čiji je naslov `"password"` (koji se prikazuje kružićima) u kolonu 1, red 2;
- programsko dugme `login` čiji je naslov `"Change"` u kolonu 1, red 3;
- reakcija na ovo dugme je zatvaranje pozornice `MyDialog`;

Za programsko dugme `login` se u kontejner `gridpane` postavlja sa desnim horizontalnim poravnanjem (`setHalignment(login, HPos.RIGHT)`). Na kraju se u `root` dodaje gridpane. Videti sliku 5.91.

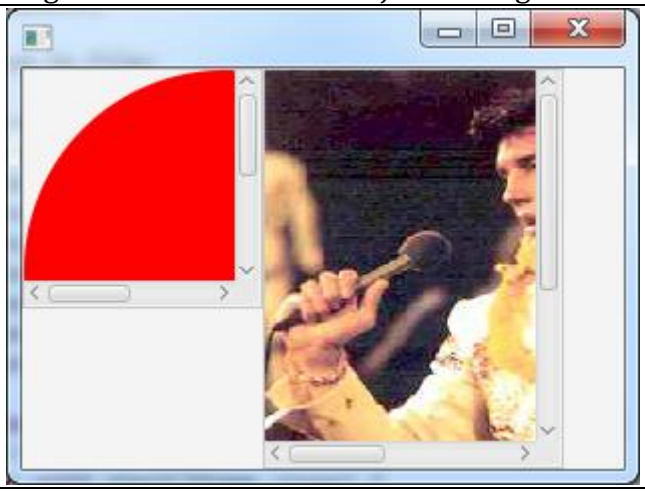
```
import javafx.geometry.HPos;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;import
javafx.scene.paint.Color;
import javafx.stage.Stage;
class MyDialog extends Stage {
    public MyDialog(Stage owner) {
        super();
        initOwner(owner);
        setTitle("title");
        setOpacity(.80);
        Group root = new Group();
        Scene scene = new Scene(root, 250, 150,Color.WHITE);
        setScene(scene);
        GridPane gridpane = new GridPane();
        gridpane.setPadding(new Insets(5));
        gridpane.setHgap(5);
        gridpane.setVgap(5);
        Label userNameLbl = new Label("User Name: ");
        gridpane.add(userNameLbl, 0, 1);

        Label passwordLbl = new Label("Password: ");
        gridpane.add(passwordLbl, 0, 2);
        final TextField userNameFld =
                                new TextField("Admin");
        gridpane.add(userNameFld, 1, 1);
        final PasswordField passwordFld =
                                new PasswordField();
        passwordFld.setText("password");
        gridpane.add(passwordFld, 1, 2);
        Button login = new Button("Change");
        login.setOnAction(e-> {close();});
        gridpane.add(login, 1, 3);
        GridPane.setHalignment(login, HPos.RIGHT);
        root.getChildren().add(gridpane);
    }
}
```

Slika 5.91. Listing klase MyDialog

5.3.3.14. Klasa ScrollPane

Klasa `ScrollPane` može da posluži za skrolovanje svog sadržaja. Na slici 5.92. dat je prikaz kruga i slike. U jednom `ScrollPane` elementu nalazi se krug, a u drugom slika. I krug i slika su većih dimenzija od datog `ScrollPane` elementa.



Slika 5.92. Aplikacija koja koristi ScrollPane

Klasa `ScrollPanePrimer_1` proširuje klasu `Application`, u `main` metodi poziva `launch(args)`, a u `start` metodi prihvata se referenca na pozornicu `stage`. Scena je dimenzija 300x200 piksela i ima korenski element `root` tipa `HBox`. Na pozornicu se postavlja scena. Kreira se crveni krug `circle` radijusa 200 piksela, sa centrom u tački (200,100). Kreira se s1 tipa `ScrollPane` veličine 120x120 piksela (postavljeno sa `setMinSize(120,120)`, `setMaxSize(120,120)`). U `s1` se postavlja sadržaj `circle` (`setContent(circle)`). Kreira se `imageview` tipa `ImageView` u koji će se postaviti slika (`Image`), kao i slika `image1` tipa `Image`. Parametri konstuktora slike su string fajla koji sadrži sliku, zahtevana širina slike, zahtevana visina slike, da li će aspekt biti sačuvan i na kraju da li će prikaz biti gladak (`new File("elvis.jpg").toURI().toString(), 300,300,false,false`)). U `imageview` se postavlja slika `image1` (`setImage(image1)`). Za `scrollpane2` tipa `ScrollPane` i preferirane veličine 150x200 piksela se postavlja sadržaj (`setContent(imageview)`). Oba elementa, `s1` i `scrollpane2`, se dodaju u `root` element. Prikazuje se pozornica. Videti sliku 5.93.

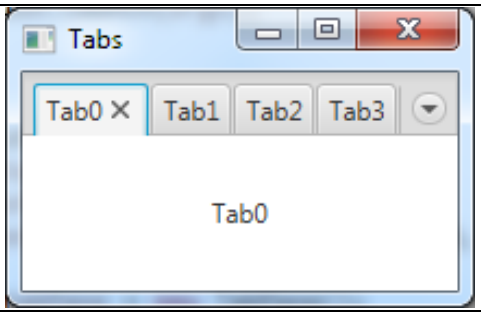
```
import java.io.File;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.ScrollPane;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
```

```
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
public class ScrollPanePrimer_1 extends Application {
  @Override public void start(Stage stage) {
    HBox root = new HBox();
    Scene scene = new Scene(root, 300, 200);
    stage.setScene(scene);
    Circle circle = new Circle(200, 200,100, Color.RED);
    ScrollPane s1 = new ScrollPane();
    s1.setMinSize(120,120);
    s1.setMaxSize(120,120); s1.setContent(circle);
    final ImageView imageview=new ImageView();
    Image image1 = new Image(
       new File("elvis.jpg").toURI().toString(),
       300,300,false,false);
    //uri,requested width,requested height,
    //preserve aspect ratio, smooth
    imageview.setImage(image1);
    ScrollPane scrolpane2 = new ScrollPane();
    scrolpane2.setPrefSize(150,200);
    scrolpane2.setContent(imageview);
    //scrollPane2.setVvalue(scrollPane2.getVmax());
    root.getChildren().addAll(s1, scrolpane2);
    stage.show();
  }
  public static void main(String[] args) {launch(args);    }  }
```

Slika 5.93. Listing aplikacije ScrollPanePrimer

5.3.3.15. Klasa TabPane

Klasa `TabPane` omogućuje kreiranje aplikacija koje imaju tabove (kartice) kao kontejnere. Videti sliku 5.94.



Slika 5.94. Izgled aplikacije koja koristi TabPane

Klasa `TabPanePrimer` proširuje klasu `Application`, u `main` metodi poziva `launch(args)`, a u `start` metodi prihvata referencu na pozornicu `primaryStage`. Postavlja se naslov pozornice "`Tabs`" i kreira root tipa Group.

Scena je dimenzija 200x100 piksela i ima korenski element `root`. Kreira se `tabPane` tipa `TabPane` i `borderPane` tipa `BorderPane`.

U programskoj petlji koja ima 5 iteracija radi se sledeće:

- kreira se `tab` tipa `Tab`;
- postavlja se naslov za `tab` (`setText(„Tab"+i)` tj. `Tab0 ... Tab4`);
- kreira se `hbox` tipa `HBox` u koji se postavlja anonimna labela čiji naslov odgovara naslovu `tab` elementa;
- postavlja se centralno poravnanje za `hbox`;
- u `tab` se postavlja `hbox` (`tab.setContent(hbox)`);
- u `tabPane` se dodaje `tab`.

Svojstvo preferirane širine i preferirane visine `borderPane` elementa vezuje se za svojstvo širine i visine scene, respektivno (npr. `borderPane.prefHeightProperty().bind(scene.heightProperty())`). Sada se `tabPane` postavlja u centar `borderPane` elementa. U `root` se dodaje `borderPane`, postavlja se scena na pozornicu i prikazuje pozornica. Videti sliku 5.95.

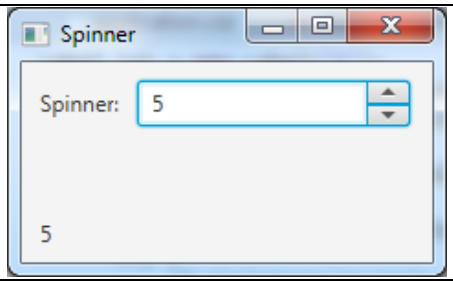
```
import javafx.application.Application; import javafx.geometry.Pos;
import javafx.scene.Group; import javafx.scene.Scene;
import javafx.scene.control.Label; import
javafx.scene.control.Tab;
import javafx.scene.control.TabPane;
import javafx.scene.layout.BorderPane; import
javafx.scene.layout.HBox;
import javafx.scene.paint.Color; import javafx.stage.Stage;
public class TabPanePrimer extends Application {
  public static void main(String[] args) { launch(args); }
  @Override public void start(Stage primaryStage) {
    primaryStage.setTitle("Tabs");
    Group root = new Group();
    Scene scene=new Scene(root,200,100,Color.WHITE);
    TabPane tabPane = new TabPane();
    BorderPane borderPane = new BorderPane();
    for (int i = 0; i < 5; i++) {
      Tab tab=new Tab();
      tab.setText("Tab"+i);
      HBox hbox = new HBox();
      hbox.getChildren().add(new Label("Tab" + i));
      hbox.setAlignment(Pos.CENTER);
      tab.setContent(hbox);
      tabPane.getTabs().add(tab);
    }
    borderPane.prefHeightProperty().
                                bind(scene.heightProperty());
```

```
        borderPane.prefWidthProperty().
                                    bind(scene.widthProperty());
        borderPane.setCenter(tabPane);
        root.getChildren().add(borderPane);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

Slika 5.95. Listing klase TabPanePrimer

5.3.3.16. Klasa Spinner

Klasa `Spinner` omogućuje izbor jednog od ponuđenih elemenata koji se korišćenjem strelica smenjuju u polju prikaza. Videti sliku 5.96.



Slika 5.96. Korišćenje klase Spinner

Klasa `SpinnerPrimer` proširuje klasu `Application`, u main metodi poziva `launch(args)`, a u `start` metodi prihvata referencu na pozornicu `stage`. Kreira se labela `lab` čiji je naslov `"5"` te `spinner` tipa `Spinner`. Pošto je ideja da skup ponuđenih elemenata bude tipa `Integer` od 0 do 10 iskorišćena je metoda `setValueFactory` da bi se u `spinner` dodale navedene vrednosti. Kao parametar ove metode kreirana je referenca na tip `SpinnerValueFactory`, a onda je pozvana njegova metoda `IntegerSpinnerValueFactory(0,10)`. Postavlja se inicijalna ponuđena vrednost pozivom `spinner.getValueFactory().setValue(Integer.parseInt("5"))`, a radilo bi i sa `setValue(5)`. Dodavanje osluškivača koji reaguje na promenu selektovane vrednosti `spinner` elementa realizovano je tako da se uzima `spinner.getEditor().textProperty()`, a onda se poziva metoda ovog svojstva `addListener`. Metoda `addListener` kao parametar ima lambda izraz `(observable, oldValue, newValue) -> { lab.setText(""+newValue);}` čime je postignuto da se nova izabrana vrednost prikaže u labeli `lab`. Kreira se `grid` tipa `GridPane`, sa vertikalnim i horizontalnim razmakom komponenti od 10 piksela i sa odstojanjem od ostalih elemenata od 10 piksela (`Insets(10)`) za sve četiri strane. U `grid` se dodaju:

- nova labela čiji je naslov `"Spinner:"` na kolonu 0 i red 0;
- `spinner` na kolonu 1 i red 0;

- `lab` na kolonu 0 i red 5

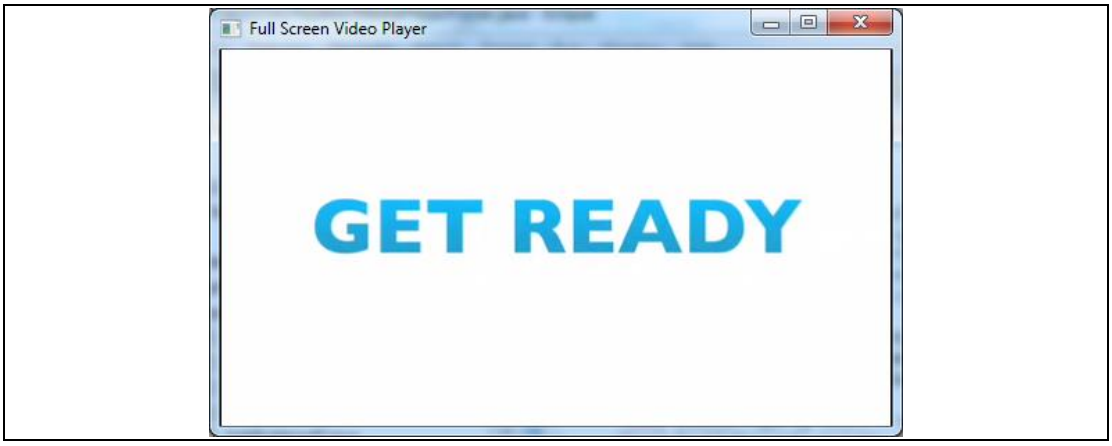
Kreira se scena sa parametrima (grid, 350,300), postavlja se naslov pozornice, scena se postavlja na pozornicu i pozornica se prikazuje. Videti sliku 5.97.

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.Spinner;
import javafx.scene.control.SpinnerValueFactory;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
public class SpinnerPrimer extends Application {
  public static void main(String[] args) { launch(args); }
    @Override public void start(Stage stage) {
    Label lab = new Label("5");
    final Spinner spinner = new Spinner();
    spinner.setValueFactory(
      new SpinnerValueFactory.IntegerSpinnerValueFactory(
                                                    0, 10));
    spinner.getValueFactory().setValue(
                                      Integer.parseInt("5"));
    //ili setValue(5)
    spinner.getEditor().textProperty().addListener(
      (observable, oldValue, newValue) -> {
                                lab.setText(""+newValue);});
    GridPane grid = new GridPane();
    grid.setHgap(10);
    grid.setVgap(10);
    grid.setPadding(new Insets(10));
    grid.add(new Label("Spinner:"), 0, 0);
    grid.add(spinner, 1, 0);
    grid.add(lab, 0, 5);
    Scene scene = new Scene(grid, 350, 300);
    stage.setTitle("Spinner");
    stage.setScene(scene);
    stage.show();
  }
}
```

Slika 5.97. Listing klase SpinnerPrimer

5.3.3.17. Prikaz multimedijalnih sadržaja

JavaFX omogućuje i prikaz multimedijalnih sadržaja. Neka je zadatak da se napravi aplikacija koja prikazuje video (tipa mp4). Videti sliku 5.98.



Slika 5.98. Pokretanje video fajla

Klasa `MediaPlayerPrimer` proširuje klasu `Application`, u `main` metodi poziva `launch(args)`, a u `start` metodi prihvata referencu na pozornicu `primaryStage`. U metodi `start` kreira se `f` tipa `File` gde se u konstruktoru prosleđuje parametar koji predstavlja stazu do fajla `trailer.mp4`. Kreira se `m` tipa `Media` čijem konstruktoru se prosleđuje parametar `f.toURI().toString()` (daće URI format `file:/C:/video/trailer.mp4`). Sada je potrebno kreirati mp tipa `MediaPlayer` čijem konstruktoru se prosleđuje parametar `m`. Ovaj plejer mp se prosleđuje konstruktoru klase `MediaView` čija je referenca `mv`. Kreiraju se dva svojstva `width` i `height` tipa `DoubleProperty` koji predstavljaju adekvatna svojstva elementa `mv` (`mv.fitWidthProperty()`, `mv.fitHeightProperty()`). Ova svojstva se vežu za adekvatna svojstva `mv.sceneProperty()` pozivima:

- `width.bind (Bindings.selectDouble(mv.sceneProperty(), "width"))`
- `height.bind(Bindings.selectDouble(mv.sceneProperty(),"height"))`

Za `mv` se postavlja očuvanje aspekta. Korenski element `root` je tipa `StackPane` i u njega se dodaje `mv`. Kreira se scena sa parametrima (`root, 960, 540`) koja se popunjava crnom bojom. Scena se postavlja na pozornicu, postavlja se naslov pozornice, postavlja se prikaz na celom ekranu (`setFullScreen(true)`), prikazuje se pozornica i startuje film (`mp.play()`). Videti sliku 5.99.

```
import java.io.File;
import javafx.application.Application;
import javafx.beans.binding.Bindings;
import javafx.beans.property.DoubleProperty;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
```

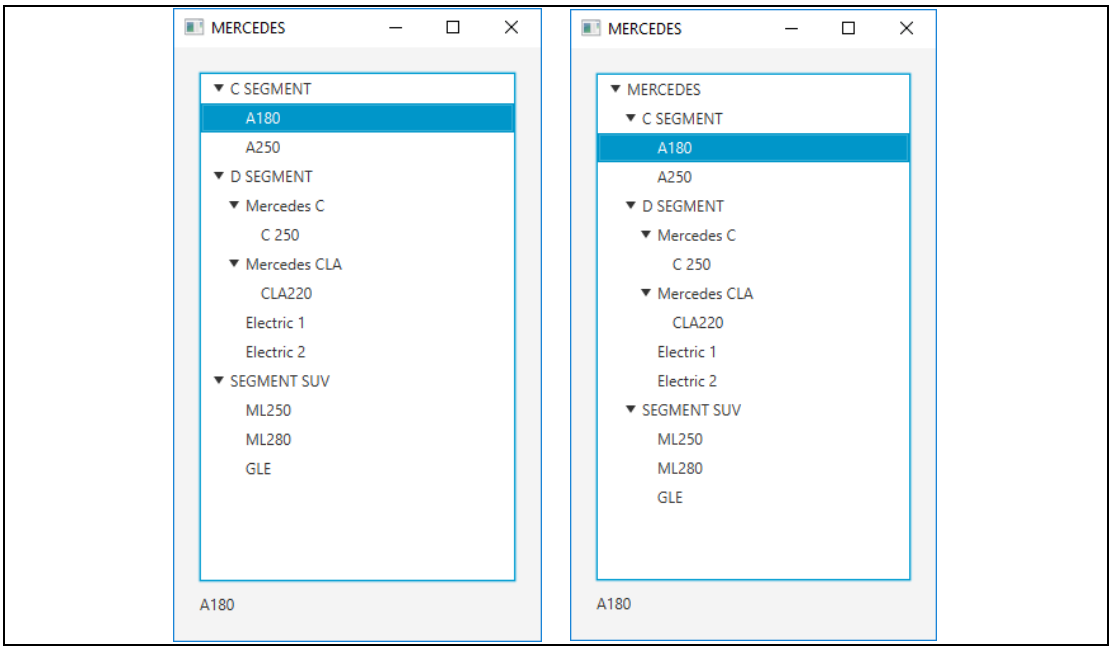
```
public class MediaPlayerPrimer extends Application {
  public static void main(String[] args) { launch(args);  }
  @Override public void start(Stage primaryStage) {
    final File f = new
             File("C:/video/trailer.mp4");
    final Media m = new Media(f.toURI().toString());
    final MediaPlayer mp = new MediaPlayer(m);
    final MediaView mv = new MediaView(mp);
    final DoubleProperty width = mv.fitWidthProperty();
    final DoubleProperty height = mv.fitHeightProperty();
    width.bind(Bindings.selectDouble(
                            mv.sceneProperty(), "width"));
    height.bind(Bindings.selectDouble(
                           mv.sceneProperty(), "height"));
    mv.setPreserveRatio(true);
    StackPane root = new StackPane();
    root.getChildren().add(mv);
    final Scene scene = new Scene(root, 960, 540);
    scene.setFill(Color.BLACK);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Full Screen Video Player");
    primaryStage.setFullScreen(true);
    primaryStage.show();
    mp.play();
  }
}
```

Slika 5.99. Kod aplikacije za pokretanje video fajla

5.3.3.18. Klasa TreeView

Prikaz komponenata koje su logički hijerarhijski povezane može biti u obliku stabla. Na slici 5.100. dato je stablo koje sadrži neka vozila marke `Mercedes` koja su raspoređena prema kategoriji kojoj pripadaju. Klasa `Mercedes` proširuje klasu `Application`, u `main` metodi poziva `launch(args)`, a u `start` metodi prihvata referencu na pozornicu `primaryStage`. Ova klasa ima dva atributa: referencu na prikaz stabla koje prikazuje stringove (`TreeView<String> tree`) i referencu na labelu u koju će se upisivati selektovani element stabla (`Label lblShowName`). U metodi start deklariše se 6 referenci tipa `TreeItem<String>`: `root`, `csegment`, `bsegment`, `suvsegment`, `mercedesc` i `mercedese`. Definiše se korenska reference sa `root=new TreeItem<String>("MERCEDES")`. Postavlja se da se `root` element prikazuje razgranato pozivom metode `root.setExpanded(true)`. Referenca `csegment` se definiše kao povratna vrednost metode `makeShow("C SEGMENT", root)` čime `csegment` postaje čvor gde počinje grana stabla. Metoda `makeShow` kreira referencu `show` na tip `TreeItem<String>` čijem konstruktoru se prosleđuje naslov koji odgovara

pravom parametru metode `makeShow` (u primeru je to `"C SEGMENT"`). Za show se postavlja da je inicijalno razgranato, a onda se show dodaje kao dete drugog prosleđenog parametra metode `makeShow` (u primeru je to `root`). Na kraju se kao povratna vrednost vraća referenca `show`. Nadalje se kreiraju i ostali delovi stabla stabla da bi se dobila hijerarhija koja je data na slici 5.100. Ako se metoda `makeShow` samo poziva bez uzimanja njene povratne vrednosti, onda se radi o kreiranju lista na stablu (npr. `makeShow("A180", csegment)`).



Slika 5.100. Stablo bez prikaza korenskog elementa (levo) i sa prikazom korenskog elementa (desno)

Definiše se referenca `tree` gde se konstruktoru prikaza stabla prosleđuje parametar `root` kao `TreeItem<String>`. Postavlja se da stablo prikazuje i korenski čvor (`setShowRoot(true)`). Dodavanje osluškivača za člana stabla se radi pozivom `tree.getSelectionModel().selectedItemProperty().addListener`. Parametar poziva metode `addListener` je lambda izraz u kome su parametri (`v, oldValue, newValue`), a metoda koja se poziva je `tree_SelectionChanged(newValue)`.

Definiše se referenca `lblShowName` te referenca `pane` na `VBox` inicijalnog razmaka među komponentama od 10 piksela i granicama prema drugim komponentama 20,20,20,20. U `pane` se dodaju `tree` i `lblShowName`, kreira se scena koja se postavlja na pozornicu, postavlja naslov pozornice i ista prikazuje.

Metoda `tree_SelectionChanged` prihvata referencu `item` na `TreeItem<String>` i ako je ista različita od `null`, onda se na labelu `lblShowName` postavlja tekst koji

odgovara elementu stabla koji je selektovan (`setText(item.getValue())`). Videti sliku 5.101.

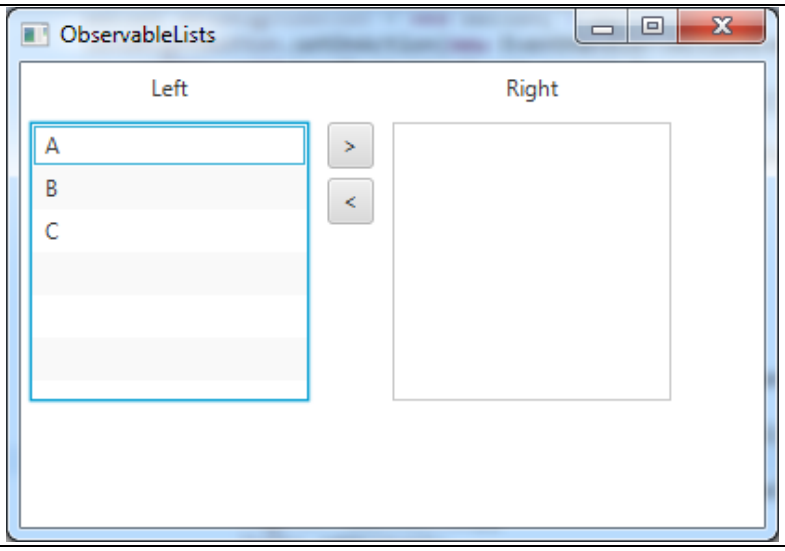
```
import javafx.application.*;
import javafx.stage.*;
import javafx.scene.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
import javafx.geometry.*;
public class Mercedes extends Application {
    public static void main(String[] args){ launch(args); }
    TreeView<String> tree;
    Label lblShowName;
    @Override public void start(Stage primaryStage){
        TreeItem<String>  root, csegment, bsegment,
                suvsegment, mercedesc, mercedese;
        root = new TreeItem<String>("MERCEDES");
        root.setExpanded(true);
          csegment = makeShow("C SEGMENT", root);
             makeShow("A180", csegment);
             makeShow("A250", csegment);
          bsegment = makeShow("D SEGMENT", root);
             mercedesc = makeShow("Mercedes C", bsegment);
                makeShow("C 250", mercedesc);
             mercedese = makeShow("Mercedes CLA", bsegment);
                makeShow("CLA220", mercedese);
             makeShow("Electric 1", bsegment);
             makeShow("Electric 2", bsegment);
          suvsegment = makeShow("SEGMENT SUV", root);
             makeShow("ML250", suvsegment);
             makeShow("ML280", suvsegment);
             makeShow("GLE", suvsegment);
        tree = new TreeView<String>(root);
        tree.setShowRoot(true);
        tree.getSelectionModel().selectedItemProperty()
            .addListener( (v, oldValue, newValue) ->
                tree_SelectionChanged(newValue) );
        lblShowName = new Label();
        VBox pane = new VBox(10);
        pane.setPadding(new Insets(20,20,20,20));
        pane.getChildren().addAll(tree, lblShowName);
        Scene scene = new Scene(pane);
        primaryStage.setScene(scene);
        primaryStage.setTitle("MERCEDES");
        primaryStage.show();
```

```
    }
    public TreeItem<String> makeShow(
                    String title,  TreeItem<String> parent){
        TreeItem<String> show = new TreeItem<String>(title);
        show.setExpanded(true);
        parent.getChildren().add(show);
        return show;
    }
    public void tree_SelectionChanged(TreeItem<String> item)
    {
        if (item != null){
            lblShowName.setText(item.getValue());
        }
    }
}
```

Slika 5.101. Aplikacija za prikaz stabla koje sadrži neke automobile marke Mercedes koji su raspoređeni po kategorijama

5.3.3.19. Klasa ListView

Ideja je da se kreiraju leva i desna lista i dva programska dugmeta koji omogućuju da se selektovani element jedne liste prebaci u drugu listu. Videti sliku 5.102.



Slika 5.102. Aplikacija za prebacivanje elemenata dve liste

Klasa `ListViewPrimer` proširuje klasu `Application`, u `main` metodi poziva `Application.launch(args)`, a u `start` metodi prihvata referencu na pozornicu `primaryStage`.

U `start` metodi postavlja se naziv pozornice i definiše referenca `root` na objekat tipa `Group`. Kreira se scena, bele pozadine, veličine 400x250 piksela sa korenskim elementom `root`. Definiše se referenca gridpane na objekat tipa `GridPane`. Postavlja se razmak od okolnih elemenata od 5 piskela za sve četiri strane te horizontalni i vertikalni razmak među komponentama od 10 piksela. Kreira se labela `candidatesLbl` čiji je naslov `"Left"` i postavlja se centralno horizontalno poravnanje za ovu labelu (`GridPane.setHalignment( candidatesLbl, HPos.CENTER)`). Navedena labela se dodaje u `gridpane` u kolonu 0 i red 0. Slično je urađeno i sa labelom `heroesLbl` čiji je naslov `"Right"`, koja se postavlja centralno u `gridpane` u kolonu 2 i red 0.

Definiše se lista `lefts` tipa referenca na `ObservableList<String>` i popunjava stringovima `"A"`, `"B"` i `"C"`. Kreira se prikaz leve liste `ListView<String> leftListView` koja kao konstruktorski argument prihvata `lefts`. Postavlja se veličina za prikaz leve liste na 150x150 (`setPrefWidth(150)`, `setPrefHeight(150)`). Prikaz liste se postavlja u `gridpane` u kolonu 0 i red 1.

Isto kao za levu listu i prikaz leve liste postavlja se desna lista i prikaz desne liste. Prikaz desne liste se postavlja u `gridpane` na kolonu 2 i red 1.

Definiše se referenca `sendRightButton` na objekat tipa `Button` čiji je naslov `">"`. Postavlja se rekcija na događaj klika na ovo programsko dugme pozivom metode `setOnAction`. Parametar ove metode je referenca na anonimnu klasu koja implementira `EventHandler<ActionEvent>` tako što definiše metodu `handle` koja prihvata referencu na `ActionEvent`. U `handle` metodi se uzima selektovani string iz prikaza leve liste ili `null` ako nema selekcije pozivom `leftListView.getSelectionModel().getSelectedItem()`. Ako ovaj string nije `null`, onda se iz `lefts` uklanja selektovani element (`lefts.remove(item)`) i dodaje u `rights` (`rights.add(item)`). Suprotno vredi za programsko dugme `sendLeftButton`.

Kreira se `VBox` u koji se stavljaju pomenuta dva programska dugmeta. Ovaj `VBox` se stavlja u `gridpane` na kolonu 1 i red 1. U `root` se dodaje `gridpane`, scena se postavlja na pozornicu i ista prikazuje. Videti sliku 5.103.

```
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.HPos;
import javafx.geometry.Insets;
import javafx.geometry.VPos;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
```

```
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
public class ListViewPrimer extends Application {
    public static void main(String[] args) {
        Application.launch(args);
    }
    @Override public void start(Stage primaryStage) {
        primaryStage.setTitle("ObservableLists");
        Group root = new Group();
        Scene scene = new Scene(root, 400, 250,Color.WHITE);
        GridPane gridpane = new GridPane();
        gridpane.setPadding(new Insets(5));
        gridpane.setHgap(10);
        gridpane.setVgap(10);
        Label candidatesLbl = new Label("Left");
        GridPane.setHalignment(candidatesLbl, HPos.CENTER);
        gridpane.add(candidatesLbl, 0, 0);
        Label heroesLbl = new Label("Right");
        gridpane.add(heroesLbl, 2, 0);
        GridPane.setHalignment(heroesLbl, HPos.CENTER);
        final ObservableList<String> lefts =
             FXCollections.observableArrayList("A","B","C");
        final ListView<String> leftListView =
                        new ListView<String>(lefts);
        leftListView.setPrefWidth(150);
        leftListView.setPrefHeight(150);
        gridpane.add(leftListView, 0, 1);
        final ObservableList<String> rights =
                         FXCollections.observableArrayList();
        final ListView<String> rightListView =
                             new ListView<String>(rights);
        rightListView.setPrefWidth(150);
        rightListView.setPrefHeight(150);
        gridpane.add(rightListView, 2, 1);
        Button sendRightButton = new Button(">");
        sendRightButton.setOnAction(
          new EventHandler<ActionEvent>() {
            public void handle(ActionEvent event) {
                String item =
                  leftListView.getSelectionModel().
                                          getSelectedItem();
                if (item != null) {
                    leftListView.getSelectionModel().
```

```
                                        clearSelection();
                    lefts.remove(item);
                    rights.add(item);
                }
            }
        });
        Button sendLeftButton = new Button("<");
        sendLeftButton.setOnAction(
          new EventHandler<ActionEvent>() {
            public void handle(ActionEvent event) {
                String item =
                rightListView.getSelectionModel().
                                        getSelectedItem();
            if (item != null) {
                    rightListView.getSelectionModel().
                                        clearSelection();
                    rights.remove(item);
                    lefts.add(item);
                }
            }
        });
        VBox vbox = new VBox(5);
        vbox.getChildren().
                    addAll(sendRightButton,sendLeftButton);
        gridpane.add(vbox, 1, 1);
        root.getChildren().add(gridpane);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

Slika 5.103. Listing aplikacije ListViewPrimer

5.3.3.20. Klasa TableView

Za objašnjenje korišćenja klase `TableView` napraviće se tabela koja sadrži informacije o filmovima. Film ima atribute: naslov, godina (objavljivanja) i cena. Videti sliku 5.104.

Slika 5.104. TableView

U klasi `Movie` navedeni su privatni atributi objekta: referenca na `String` `title`, `year` tipa `int` i `price` tipa `double`. Prazan konstruktor postavja stanje objekta tako da `title`, `year` i `price` imaju vrednosti "", 0, 0.0, respektivno. Naveden je i konstruktor koji prihvata tri argumenta za postavljanje stanja objekta. Slede geteri i seteri atributa objekta ove klase. Videti sliku 5.105.

```
public class Movie {
    private String title;
    private int year;
    private double price;

    public Movie()    {
        this.title = "";
        this.year = 0;
        this.price = 0.0;
    }

    public Movie(String title, int year, double price) {
        this.title = title;
        this.year = year;
        this.price = price;
    }

    public String getTitle()           {return this.title; }
    public void setTitle(String title) {this.title = title;}
    public int getYear()               {return this.year;  }
    public void setYear(int year)      { this.year = year; }
    public double getPrice()           {return this.price; }
    public void setPrice(double price) {this.price = price;}
}
```

Slika 5.105. Listing klase Movie

Klasa MovieInventory proširuje klasu Application, u main metodi poziva launch(args), a u start metodi prihvata referencu na pozornicu primaryStage. Kreira se labela lblHeading čiji je naslov "Movie Inventory" i čiji je font Arial veličine 20 piksela. Definiše se referenca table na objekat tipa TableView<Movie>. Generik Movie znači da jedan red tabele predstavlja zapis (svih atributa) jednog filma, a samim tim jedna kolona tabele se donosi na jedan atribut u klasi Movie. Poziva se metoda setItems za popunjavanje prikaza tabele čiji je patametar poziv metode loadData(). Definiše se referenca colTitle na objekat tipa TableColumn<Movie, String> gde se konstruktoru prosle]uje naslov kolone ("Title"). Postavlja se minimalna širina kolone (setMinWidth(300)). Pozivom metode colTitle.setCellValueFactory(new PropertyValueFactory<Movie,String>("title")) postavljena je veza vrednosti ćelije kolone "Title" i atributa objekta klase Movie tipa String. U koloni naziva "Title" biće prikaz atributa objekta klase Movie čiji je identifikator "title". Na isti način kreirane su kolone tabele za prikaz godine tipa Integer naziva "Year" i cene tipa double naziva "Price" čije su minimalne širine 100 piksela. Ćelije ovih kolona imaće vrednosti atributa objekta klase Movie year i price, respektivno. Pozivom table.getColumns().addAll (colTitle, colYear, colPrice) dodaju se kolone tabele. Kreira se VBox gde je razmak među elementima 10 piksela i okolna granica prema drugim elementima 10,10,10,10 piksela. U ovaj element se dodaju lblHeading i table. Kreira se scena koja sadrži navedeni HBox, koja se postavlja na pozornicu naslova "Movie Inventory" i na kraju se prikazuje pozornica.

Metoda loadData vraća ObservableList<Movie>. U metodi je definisana referenca data pozivom metode FXCollections.observableArrayList(), a onda sledi dodavanje referenci na objekte klase Movie u listu data. Na kraju se kao povratna vrednost metode navodi lista data. Videti sliku 5.106.

```
import javafx.application.*;
import javafx.stage.*;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.scene.text.*;
import javafx.scene.control.cell.*;
import javafx.collections.*;
import javafx.geometry.*;
public class MovieInventory extends Application {
  public static void main(String[] args) { launch(args); }
  @Override
  public void start(Stage primaryStage) {
    Label lblHeading = new Label("Movie Inventory");
    lblHeading.setFont(new Font("Arial", 20));
    TableView<Movie> table = new TableView<Movie>();
```

```
    table.setItems(loadData());
    TableColumn<Movie, String> colTitle =
        new TableColumn("Title");
    colTitle.setMinWidth(300);
    colTitle.setCellValueFactory(
        new PropertyValueFactory<Movie,String>("title"));
    TableColumn<Movie, Integer> colYear =
        new TableColumn("Year");
    colYear.setMinWidth(100);
    colYear.setCellValueFactory(
        new PropertyValueFactory<Movie, Integer>("year"));
    TableColumn<Movie, Double> colPrice =
        new TableColumn("Price");
    colPrice.setMinWidth(100);
    colPrice.setCellValueFactory(
        new PropertyValueFactory<Movie, Double>("price"));
    table.getColumns().addAll(colTitle, colYear, colPrice);
    VBox paneMain = new VBox();
    paneMain.setSpacing(10);
    paneMain.setPadding(new Insets(10, 10, 10, 10));
    paneMain.getChildren().addAll(lblHeading, table);
    Scene scene = new Scene(paneMain);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Movie Inventory");
    primaryStage.show();
  }
  public ObservableList<Movie> loadData() {
    ObservableList<Movie> data =
                        FXCollections.observableArrayList();
    data.add(new Movie("It's a Wonderful Life",1946,14.95));
    data.add(new Movie("Young Frankenstein",   1974,16.95));
    data.add(new Movie("Star Wars Episode 4",  1976,17.95));
    data.add(new Movie("Glory",                1989,14.95));
    data.add(new Movie(
                       "The Invention of Lying",2009,18.95));
    data.add(new Movie("The King's Speech",    2010,19.95));
    return data;
  }
}
```

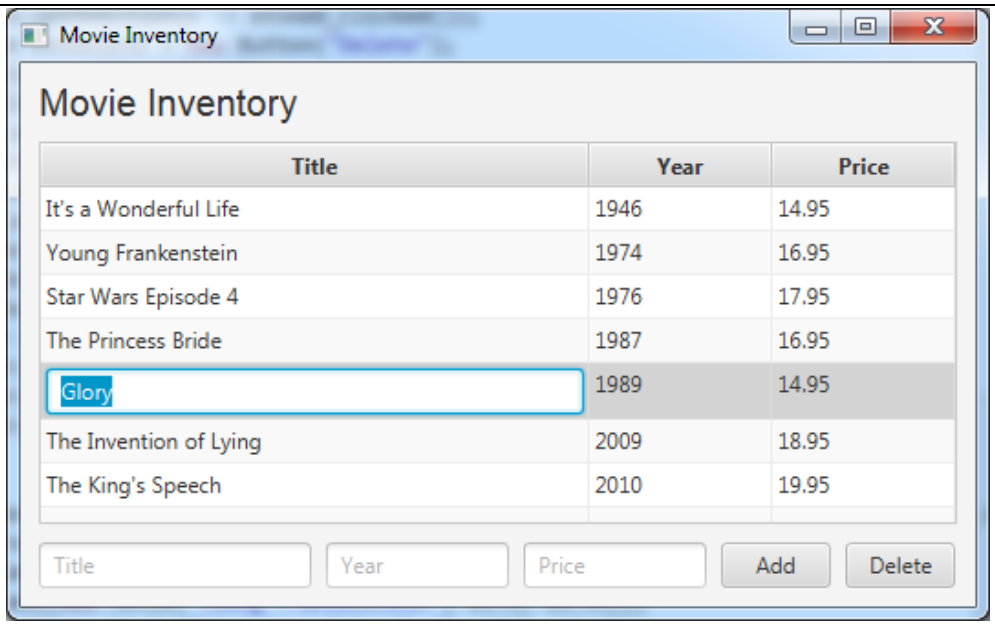
Slika 5.106. Listing klase MovieInventory

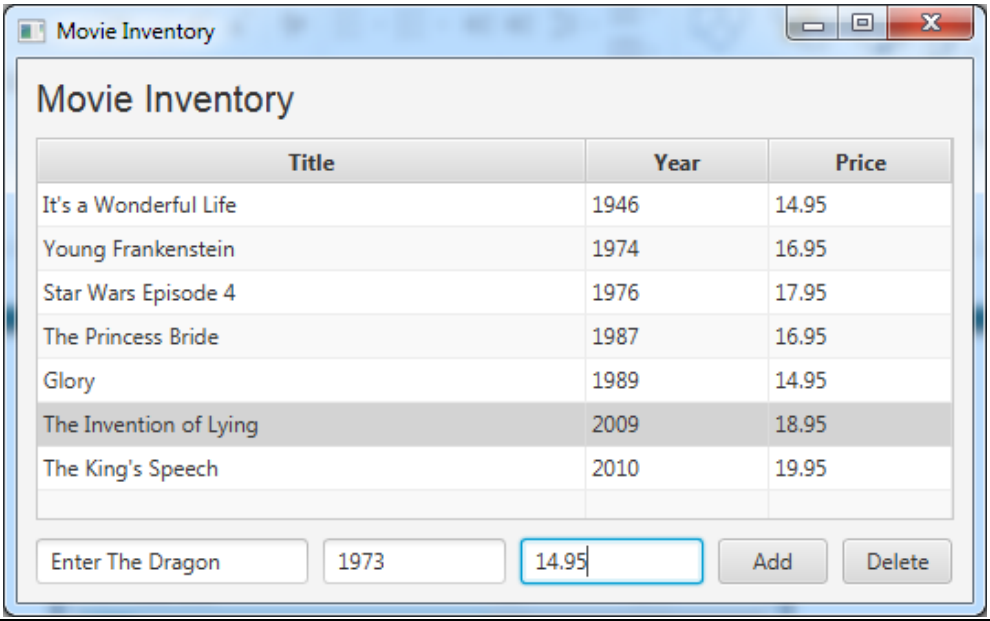
5.3.3.21. Editovanje, dodavanje I brisanje stavke tabele

Prethodni kod je moguće modifikovati tako da se može izvršiti editovanje, dodavanje i brisanje filma. Videti sliku 5.107. Programski kod za ovakvo rešenje dat je na slici 5.108. Kao i ranije, za film se koristi klasa `Movie` (opisana ranije).

Razlike koje se tiču kolone tabele (`TableColumn`) je da je dodat poziv metode `setCellFactory`. Za kolonu `colTitle` taj poziv je `colTitle.setCellFactory( TextFieldTableCell.forTableColumn())` koji će omogućiti stvaranje pripadnog polja za tekst (`TextField`) za datu ćeliju. Ovde poziv metode `forTableColumn()` nema argument jer se radi o podatku tipa `String` (naziv filma, atribut `title` u klasi `Movie`) tako da nije potrebna konverzija u `String`. Poziv metode `colTitle.setOnEditCommit(e -> colTitle_OnEditCommit(e))` omogućuje da se napuštanjem editovane ćelije pozove metoda `colTitle_OnEditCommit` kojoj se prosleđuje `Event e`.

Za kolonu `colYear` dodatni poziv u odnosu na raniji kod je `colYear.setCellFactory(TextFieldTableCell.forTableColumn(new IntegerStringConverter()))`. Ovde je potrebno metodi `forTableColumn` proslediti referencu na objekat tipa `IntegerStringConverter` pošto je `colYear` vezan za `year` atrribut klase `Movie` koji je tipa `int`. Poziv metode `colYear.setOnEditCommit(e -> colYear_OnEditCommit(e))` omogućuje da se napuštanjem editovane ćelije pozove metoda `colYear_OnEditCommit` kojoj se prosleđuje `Event e`.





Slika 5.107. Aplikacija za editovanje, dodavanje i brisanje stavke tabele

Za kolonu `colPrice` dodatni poziv u odnosu na raniji kod je `colPrice.setCellFactory(TextFieldTableCell.forTableColumn(new DoubleStringConverter()))`. Ovde je potrebno metodi `forTableColumn` proslediti referencu na objekat tipa `DoubleStringConverter` pošto je `colPrice` vezan za `price` atrribut klase `Movie` koji je tipa `double`. Poziv metode `colPrice.setOnEditCommit(e -> colPrice_OnEditCommit(e))` omogućuje da se napuštanjem editovane ćelije pozove metoda `colPrice_OnEditCommit` kojoj se prosleđuje `Event e`.

Dodaju se tri tekstualna polja: `txtTitle`, `txtYear` i `txtPrice` te dva programska dugmeta: `btnAdd` i `btnDelete`. Navedena tekstualna polja služe za upis atributa filma koji će biti dodat u tabelu filmova akcijom na programsko dugme `btnAdd`. Programsko dugme `btnDelete` svojom akcijom briše selektovani film iz tabele filmova. Postavlja se akcija na programsko dugme `btnAdd` koju obavlja metoda `btnAdd_Clicked()`, odnosno, za programsko dugme `btnDelete` postavlja se akcija koju obavlja metoda `btnDelete_Clicked()`. Ovi elementi se postavljaju u `HBox paneAdd`.

Za razliku od ranije verzije ovde se u `VBox` postavlja na kraju i `paneAdd`.

Dodatnih pet metoda u ovoj verziji koda koje obrađuju događaje su:

- `public void colTitle_OnEditCommit (Event e)`
  Kreira se referenca na događaj editovanja ćelije tabele i njoj se pridružuje prosleđeni argument `e` koji je referenca na objekat tipa `Event`. Kastovanje se vrši prema zapisu tabele i tipu atributa koji je vezan za ćeliju `(TableColumn.CellEditEvent<Movie, String> ce = (TableColumn.`

CellEditEvent<Movie, String>) e). Uzima se referenca na zapis reda tabele u kome se nalazi editovana ćelija (Movie m = ce.getRowValue()). Preko ove reference se postavlja nova vrednost ćelije kao vrednost atributa title (m.setTitle(ce.getNewValue())).

- public void colYear_OnEditCommit (Event e)
  Kreira se referenca na događaj editovanja ćelije tabele i njoj se pridružuje prosleđeni argument e koji je referenca na objekat tipa Event. Kastovanje se vrši prema zapisu tabele i tipu atributa koji je vezan za ćeliju (TableColumn.CellEditEvent<Movie, Integer> ce = (TableColumn.CellEditEvent<Movie, Integer>) e). Uzima se referenca na zapis reda tabele u kome se nalazi editovana ćelija (Movie m = ce.getRowValue()). Preko ove reference se postavlja nova vrednost ćelije kao vrednost atributa year (m.setYear(ce.getNewValue())).
- public void colPrice_OnEditCommit (Event e)
  Kreira se referenca na događaj editovanja ćelije tabele i njoj se pridružuje prosleđeni argument e koji je referenca na objekat tipa Event. Kastovanje se vrši prema zapisu tabele i tipu atributa koji je vezan za ćeliju (TableColumn.CellEditEvent<Movie, Double> ce = (TableColumn.CellEditEvent<Movie, Double>) e). Uzima se referenca na zapis reda tabele u kome se nalazi editovana ćelija (Movie m = ce.getRowValue()). Preko ove reference se postavlja nova vrednost ćelije kao vrednost atributa price (m.setPrice(ce.getNewValue())).
- public void btnAdd_Clicked()
  Definiše se referenca m na objekat tipa Movie. Postavlja se stanje ovog objekta koje odgovara sadržajima polja txtTitle, txtYear (sa parsiranjem prema tipu int) i txtPrice (sa parsiranjem prema tipu double). U tabelu se sada dodaje novi film (table.getItems().add(m). Brišu se sadržaji polja txtTitle, txtYear i txtPrice.
- public void btnDelete_Clicked()
  Deklarišu se dve reference na liste: sel i items tipa ObservableList<Movie>. Referenca items gleda na listu filmova u tabeli (tabel.getItems()). Referenca sel gleda na selektovane filmove u tabeli filmova. Iterativno se ide kroz listu selektovanih filmova (sel) i za svaki selektovani film (m) se poziva metoda koja uklanja taj film iz tabele filmova (items.remove(m)). Za višestruku selekciju zapisa u tabeli filmova koristio bi se sledeći poziv: table.getSelectionModel().setSelectionMode (SelectionMode.MULTIPLE).

```
import javafx.application.*;
import javafx.stage.*;
import javafx.scene.*;
import javafx.scene.control.*;
```

```
import javafx.scene.layout.*;
import javafx.scene.text.*;
import javafx.event.*;
import javafx.scene.control.cell.*;
import javafx.beans.property.*;
import javafx.collections.*;
import javafx.geometry.*;
import javafx.util.converter.*;
public class MovieInventoryEditor extends Application {
  public static void main(String[] args) { launch(args); }
  private TableView<Movie> table;
  private TextField txtTitle, txtYear, txtPrice;
  @Override
  public void start(Stage primaryStage) {
    Label lblHeading = new Label("Movie Inventory");
    lblHeading.setFont(new Font("Arial", 20));
    table = new TableView<Movie>();
    table.setEditable(true);
    table.setItems(loadData());
    TableColumn colTitle = new TableColumn("Title");
    colTitle.setMinWidth(300);
    colTitle.setCellValueFactory(
        new PropertyValueFactory<Movie, String>("title"));
    colTitle.setCellFactory(
        TextFieldTableCell.forTableColumn());
    colTitle.setOnEditCommit(e -> colTitle_OnEditCommit(e));
    TableColumn colYear = new TableColumn("Year");
    colYear.setMinWidth(100);
    colYear.setCellValueFactory(
        new PropertyValueFactory<Movie, Integer>("year"));
    colYear.setCellFactory(
        TextFieldTableCell.forTableColumn(
            new IntegerStringConverter()));
    colYear.setOnEditCommit(e -> colYear_OnEditCommit(e));
    TableColumn colPrice = new TableColumn("Price");
    colPrice.setMinWidth(100);
    colPrice.setCellValueFactory(
        new PropertyValueFactory<Movie, Double>("price"));
    colPrice.setCellFactory(
        TextFieldTableCell.forTableColumn(
            new DoubleStringConverter()));
    colPrice.setOnEditCommit(e -> colPrice_OnEditCommit(e));
    table.getColumns().addAll(colTitle, colYear, colPrice);
    txtTitle = new TextField();
    txtTitle.setPromptText("Title");
    txtTitle.setMinWidth(100);
    txtYear = new TextField();
```

```
    txtYear.setMaxWidth(100);
    txtYear.setPromptText("Year");
    txtPrice = new TextField();
    txtPrice.setMaxWidth(100);
    txtPrice.setPromptText("Price");
    Button btnAdd = new Button("Add");
    btnAdd.setMinWidth(60);
    btnAdd.setOnAction(e -> btnAdd_Clicked());
    Button btnDelete = new Button("Delete");
    btnDelete.setMinWidth(60);
    btnDelete.setOnAction(e -> btnDelete_Clicked());
    HBox paneAdd = new HBox();
    paneAdd.setSpacing(8);
    paneAdd.getChildren().addAll(txtTitle, txtYear,
        txtPrice, btnAdd, btnDelete);
    VBox paneMain = new VBox();
    paneMain.setSpacing(10);
    paneMain.setPadding(new Insets(10, 10, 10, 10));
    paneMain.getChildren().addAll(
                          lblHeading, table, paneAdd);
    Scene scene = new Scene(paneMain);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Movie Inventory");
    primaryStage.show();
  }
  public ObservableList<Movie> loadData() {
    ObservableList<Movie> data =
                        FXCollections.observableArrayList();
    data.add(new Movie("It's a Wonderful Life", 1946,14.95));
    data.add(new Movie("Young Frankenstein",    1974,16.95));
    data.add(new Movie("Star Wars Episode 4",   1976,17.95));
    data.add(new Movie("The Princess Bride",    1987,16.95));
    data.add(new Movie("Glory",                 1989,14.95));
    data.add(new Movie("The Invention of Lying",2009,18.95));
    data.add(new Movie("The King's Speech",     2010,19.95));
    return data;
  }
  public void colTitle_OnEditCommit(Event e) {
    TableColumn.CellEditEvent<Movie, String> ce;
    ce = (TableColumn.CellEditEvent<Movie, String>) e;
    Movie m = ce.getRowValue();    m.setTitle(ce.getNewValue());
  }
  public void colYear_OnEditCommit(Event e) {
    TableColumn.CellEditEvent<Movie, Integer> ce;
```

```
    ce = (TableColumn.CellEditEvent<Movie, Integer>) e;
    Movie m = ce.getRowValue();     m.setYear(ce.getNewValue());
  }
  public void colPrice_OnEditCommit(Event e) {
    TableColumn.CellEditEvent<Movie, Double> ce;
    ce = (TableColumn.CellEditEvent<Movie, Double>) e;
    Movie m = ce.getRowValue();     m.setPrice(ce.getNewValue());
  }
  public void btnAdd_Clicked() {
    Movie m = new Movie();
    m.setTitle(txtTitle.getText());
    m.setYear(Integer.parseInt(txtYear.getText()));
    m.setPrice(Double.parseDouble(txtPrice.getText()));
    table.getItems().add(m);
    txtTitle.clear();    txtYear.clear();    txtPrice.clear();
  }
  public void btnDelete_Clicked() {
    ObservableList<Movie> sel, items;
    items = table.getItems();
    sel = table.getSelectionModel().getSelectedItems();
    for (Movie m : sel) items.remove(m);
  }
}
```

Slika 5.108. Listing aplikacije koja omogućuje dodavanje, editovanje i brisanje stavke tabele

5.3.4. JavaFX, paralelni procesi

Za kreiranje paralelnog procesa u JavaFX tehnologiji korsite se dva pristupa: `Platform.runLater` i `Task`.

`Platform.runLater` (`javafx.application.Platform`) koristi se za kreiranje paralelnih procesa koji obavljaju manji posao. Neka se posmatra TCP komunikacija klijent-server. Da bi se na klijentskoj strani koja obavlja i komunikaciju sa serverom (GUI je realizovan primenom JavaFX tehnologije) u datoj labeli odvijalo sledeće:

- ispisivanje "SRECNO"
- pravljenje pauze
- ispisivanje "S R E C N O"
- pravljenje pauze
- ponavljanje navedene sekvence

potrebno je koristiti kao što sledi:

```
...
// kreiranje labele na kojoj piše SRECNO
   Label lab_SRECNO = new Label("SRECNO");
```

```
// postavljanje da se labela prostire na maksimalnu dužinu
   lab_SRECNO.setMaxWidth(Double.MAX_VALUE);
// pozicioniranje labele
   lab_SRECNO.setAlignment(Pos.TOP_CENTER);
// centriranje teksta labele
   lab_SRECNO.setTextAlignment(TextAlignment.CENTER);
// postavljanje stila prikaza labele
   lab_SRECNO.setStyle(
    "-fx-text-fill:red;-fx-font-weight:bold;-fx-font-size:28px;");
...
// dodavanje labele u korenski element (VBox root = new VBox(10);)
   root.getChildren().add(lab_SRECNO);
...
// kreiranje i pokretanje niti (koja obavlja mali posao)
   Thread thread = new Thread(new Runnable() {
     @Override
     public void run() {
       Runnable updater = new Runnable() {
         @Override
         public void run() {
           if(lab_SRECNO.getText().equals("SRECNO"))
                   lab_SRECNO.setText("S  R  E  C  N  O");
           else
                   lab_SRECNO.setText("SRECNO");
         }
       };
       while (true) {
         try { Thread.sleep(1000); }
         catch (InterruptedException ex) {}
         Platform.runLater(updater);
       }
     }
   });
   thread.setDaemon(true);
   thread.start();
...
```

U niti `thread` u metodi `run`:

- definisan je `updater` koji u svojoj `run` metodi proverava trenutni ispis labele da bi postavio drugi;
- u beskonačnoj petlji pravi se pauza od 1000ms, a onda poziva metoda `Platform.runLater` kojoj se prosleđuje `updater`.

Nit `thread` se postavlja kao demonska i startuje se.

`Task` (`javafx.concurrent.Task`) se koristi za kreiranje paralelnih procesa koji obavljaju veći posao. Neka je serverska strana u TCP komunikaciji realizovana

u JavaFX tehnologiji i neka se na serverskoj strani prikazuje lista u koju se ubacuju ispisi servera i poruke koje server dobija od klijenta.

```
...
public class ServerK2 extends Application {
   public static final int TCP_PORT = 9000;
   private BufferedReader in;
   private PrintWriter out;
   private final static ObservableList<String> elementiListe =
              FXCollections.observableArrayList("*** SERVER ***");
   private static ListView<String> listView;
   public static void main(String[] args) {     launch(args);   }
   @Override
   public void start(Stage primaryStage) {
     primaryStage.setTitle("SERVER");
     Group root = new Group();
     Scene scene = new Scene(root, 400, 800, Color.LIGHTYELLOW);
     listView = new ListView<String>(elementiListe);
     listView.setStyle("-fx-font-size:20px;");
     listView.setMinWidth(400);     listView.setMinHeight(800);
     VBox vbox = new VBox(5);
     vbox.getChildren().addAll(listView);
     root.getChildren().add(vbox);
     primaryStage.setScene(scene);     primaryStage.show();
     Task task = new Task<Void>() {
            @Override public Void call() {
                  idiposlestarta();
                  return null;
            }
      };
      new Thread(task).start();
   }
   public void idiposlestarta(){
     try {
       ServerSocket ss = new ServerSocket(TCP_PORT);
       String informacija="Server je pokrenut.";
       elementiListe.add(informacija);
       while (true) {
         Socket sock = ss.accept();
         elementiListe.add("Client accepted:");
         in = new BufferedReader(
                    new InputStreamReader(sock.getInputStream()));
         out = new PrintWriter( new BufferedWriter(
           new OutputStreamWriter(sock.getOutputStream())), true);
         String odgovor = "[SERVER] : ";
         String request = in.readLine();
```

```
            elementiListe.add(request);
            String[] delovi = request.split(":");
            if(delovi[0].equals("VREME")){
               Date dt = new Date();
               SimpleDateFormat sdf =
                  new SimpleDateFormat("hh:mm:ss");
               String time1 = sdf.format(dt);
               odgovor+=time1;
            }
            out.println(odgovor);
            in.close();       out.close();      sock.close();
          }
         }
         catch (Exception ex) {  ex.printStackTrace(); }
      }
}
```

`Task` je generička klasa koja u svojoj metodi `call` vraća tip koji je naveden. U primeru je korišćen tip `Void` koji u Javi predstavlja specijalan tip kome se jedino može pridružiti vrednost `null`. Kada bi bila potrebna povratna vrednost npr. tipa `Integer`, onda bi umesto `<Void >` stajao `<Integer>` i naredba `return` bi umesto `null` morala da vrati referencu na objekat tipa `Integer`. Nakon definisanja metode `call` (koja zove metodu `idiposlestarta`) i koja vraća `null`, kreirana je neimenovana nit koja se inicijalizuje referencom `task` i koja se odmah startuje. U metodi `idiposlestarta` se obavlja kompletna komunikacija servera sa klijentom.

Kompletirajte kod serverske strane i proširite ga novim funkcionalnostima !

Rezime poglavlja grafički korisnički interfejs

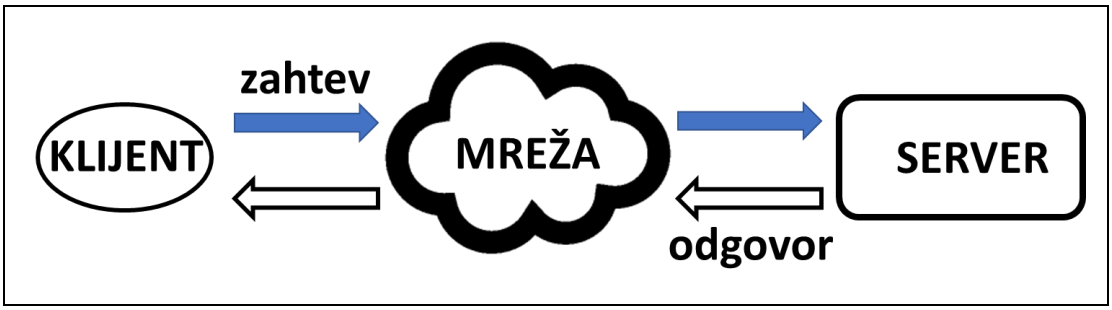
U poglavlju grafički korisnički interfejs obrađeno je programiranje grafičkih interfejsa korišćenjem tehnologija: AWT (*Abstract Windowing Toolkit*), Swing i JavaFX.

Zadaci za proveru znanja iz poglavlja GUI:

1. Napisati AWT, Swing, JavaFX program koji omogućuje unos i množenje dve matrice dimenzija 3x3, kao i prikaz rezultata množenja.
2. Napisati AWT, Swing, JavaFX program koji proverava da li su dve rečenice unesene u dva tekstualna polja anagrami. Obe rečenice su anagrami ako se premeštanjem slova jedne rečenice može dobiti druga rečenica. Razmaci se ne računaju.
   Primer: jadransko more --- krasan je odmor
   Pomoć: obe rečenice imaju isti broj i istu frekvenciju slova, a razmislite o sortiranju, trimovanju i poređenju.
3. Napisati AWT, Swing, JavaFX program koji prikazuje matricu 8x8 polja i gde se pijani moler kreće slučajno (pijano) po matrici. Moler ostavlja brojkoraka%10 (0..9) na polju koje je obojio. Kada oboji sva polja progam dojavljuje da je moler obavio posao.
4. Napisati AWT, Swing, JavaFX program koji za datu matricu 3x3 prikazuje njenu inverznu matricu (ako je ima).
5. Napisati AWT, Swing, JavaFX program koji prikazuje lavirint kao matricu karaktera. Neka je zid #, početno polje robota R i izlaz E. Hodnici lavirinta su širine jednog karaktera. Program za postavljeno početno stanje robota R ispisuje jedno rešenje kako da se robot kreće da bi došao do izlaza E.
   Primer:
   ```
   ######
   # R  #
   # ## #
   #    #
   ###E##
   ```
   Resenje je: desno-desno-dole-dole-levo-dole
   Pomoć: razmislite o rekurzivnom rešenju.

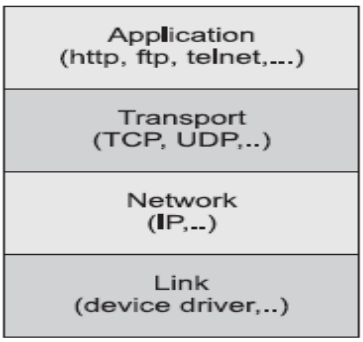
6. KLIJENT-SERVER KOMUNIKACIJA

Cilj poglavlja je da student ovlada pisanjem klijent-server programa koji komuniciraju u računarskoj mreži korišćenjem TCP (*Transport Control Protocol*) i UDP (*User Datagram Protocol*) protokola.



Slika 6.1. Komunikacija između klijenta i servera

Mrežni računarski sistemi sastoje se od servera, klijenata i komunikacione opreme. Na slici 6.1. dat je prikaz komunikacije između kliljenta i servera. Računarski program koji šalje zahtev za usluge se zove klijentski program, a računar na kom se ovaj program izvršava zove se klijentski računar. Ove zahteve prihvata, obrađuje i pruža usluge klijentima serverski program koji se izvršava na serverskom računaru. Komunikacija se svodi na slanje zahteva klijenta ka aktivnom serveru preko komunikacionih medijuma. Trivijalno, i server i klijent mogu biti pokrenuti na istom računaru. Serverski program obrađuje pristigli zahtev klijenta i vraća rezultat (odgovor) klijentu.



Slika 6.2. TCP/IP softverski stek

Klijentski i serverski programi uključuju mrežne usluge transportnog sloja, koji je deo softverskog Internet steka (*Internet software stack*) odnosno TCP/IP (Transport Control Protocol/Internet Protocol) steka koji je prikazan na slici 6.2.

Transportni sloj se sastoji od dve vrste protokola: TCP (*Transport Control Protocol*) i UDP (*User Datagram Protocol*). Najkorišćeniji programski interfejs za ove protokole je mrežna utičnica (*socket*).

TCP je konekciono orijentisan (*connection-oriented*) protokol koji obezbeđuje pouzdan tok podataka između dva računara. Primer aplikacija koje koriste takve usluge su HTTP (*HyperText Transfer Protocol*), FTP (*File Trasfer Protocol*) i Telnet. TCP garantuje isporuku paketa i očuvava njihov redosled kojim pristižu na odredište.

UDP je protokol koji šalje nezavisne pakete podataka koji se nazivaju datagrami, od jednog računara do drugog, bez garancije o dolasku i sledu dolazaka datagrama. Primer aplikacije koja koristi takve usluge je ping.

Mrežno programiranje je pisanje programa koji komuniciraju sa drugim programima (obratiti pažnju da je reč o programima) preko računarske mreže.

Da bi se pristupilo pisanju navedenih programa potrebno je prvo definisati komunikacioni protokol. Komunikacioni protokol je skup pravila koja definišu uspostavljanje veze, održavanje veze, oporavak veze u slučaju prekida i završetak veze.

Standardna Java biblioteka poseduje klase za realizaciju mrežne komunikacije bazirane na protokolima: TCP i UDP. Prednost Jave je platformska nezavisnost tako da se isti programi mogu koristiti na različitim hardverskim i softverskim platformama.

Komuniciranje između dva programa odvija se korišćenjem tokova (*streams*) pri čemu je komunikacija između ova dva programa dvosmerna. Dvosmernost je rešena korišćenjem *stream-a* za čitanje i *stream-a* za pisanje. Prvi stream prima podatke od druge strane, a drugi *stream* šalje podatke ka drugoj strani.

Stream-ovi se koriste kao što je prikazano u ranijim poglavljima samo što se razlikuje izvor odnosno odredište toka podataka.

Klase namenjene za mrežnu komunikaciju nalaze se u paketu *java.net*, a klase koje realizuju stream-ove nalaze se u paketu *java.io*.

6.1. IP adresa

Identifikator čvora u IP mreži određen je IP adresom koja predstavlja 32-bitni broj za IPv4 standard ili 128-bitni broj za IPv6 standard.

IP adresa za IPv4 je sastavljena od 4 bajta i obično se piše po vrednostima bajtova razdvojenih tačkom, na primer, 192.168.100.1 (adresa modema). Vrednosti bajta su neoznačene (opseg od 0 do 255).

IP adresa se obično mapira u naziv kao www.google.com koji je lakši za pamćenje. U okviru servisa za imenovanje domena DNS (*Domen Name System*) nalazi se sistem imena koji odgovaraju IP brojevima. Četiri broja u IP adresi za IPv4 gledano sleva nadesno određuju hijerarhiju mreže dok naziv Internet adrese, koje se zove naziv domena, opisuje lokaciju računara u imenskom

prostoru, ali gledano zdesna ulevo.

IP adresa ima deo koji predstavlja adresu IP mreže (ista za sve računare u jednoj IP mreži) i deo koji predstavlja adresu računara (jedinstvena za svaki računar u istoj IP mreži). U zavisnosti od vodeće grupe bitova, IP adrese se dele na klase prema tabeli 6.1. U decimalnoj notaciji ove klase bi male opsege: 0-127, 128-191, 192-223, 224-239, 240-255 respektivno za klase A, B, C, D i E.

Tabela 6.1. Klase IP adresa

Klasa	Vodeći bitovi	Broj bitova za mrežu	Broj bitova za host
A	0	7	24
B	10	14	16
C	110	21	8
D (multikast)	1110		
E (rezervisana)	1111		

IP adresa za IPv6 predstavljena je kao niz osam 16-bitnih heksadecimalnih brojeva sa dvotačkom (:) kao graničnikom.

Primer IPv6 adrese:

```
2056:0000:130F:0000:0000:00F0:9876:0005
```

Početne nule u svakom heksadecimalnom polju su opcione i mogu se izostaviti pri predstavljanju adrese.

Primer:

```
2056:0000:230C:0000:0000:00F0:9876:0005 =
2056:0:230C:0:0:F0:9876:5
```

Znakom dve dvotačke (::) označavaju se uzastopna polja koja se sastoje samo od 0, pri čemu se ovaj znak može upotrebiti samo jednom u IPv6 adresi.

Primer:

```
2056:0000:230C:0000:0000:00F0:9876:0005 =
2056:0:230C::00F0:9876:0005
```

6.2. Port

Često je potrebno da računar istovremeno komunicira sa više računara i da je u pitanju veći broj različitih usluga (istovremeno veći broj ftp sesija, web konekcija i chat-ova) tako da je potrebno mapiranje pristiglih paketa podataka na odgovarajući proces budući da se sve komunikacije obavljaju preko jedne IP adrese (neka računar ima jednu mrežnu karticu).

Mapiranje se realizije pomoću portova. Port predstavlja logičku pristupnu tačku za proces kom je paket podataka namenjen. Port je određen vrednošću 16-bitnog neoznačenog broja. Ovim mapiranjem svaka usluga koju nudi računar je

jedinstveno određena brojem porta na datoj IP adresi. Svaka IP adresa ima skup od 64K portova.

Svaki Internet paket sadrži i odredišnu adresu hosta i broj porta čime je određeno kom procesu će na odredišnom računaru paket biti dostavljen.

Pomenuti protokoli: TCP i UDP koriste portove za preslikavanje ulaznih podataka na određeni proces koji se izvodi na računaru. Neki portovi su rezervisani za podršku poznatih (well-known) usluga što je dato u tabeli 6.2.

Tabela 6.2. Portovi rezervisani za poznate servise

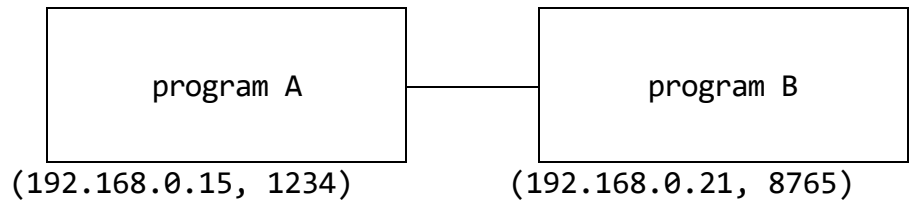
Servis	Port	Protokol
ftp (file trasfer protocol)	21	tcp
telnet	23	tcp
smtp (simple mail transfer protocol)	25	tcp
login	513	tcp
http (hypertext transfer protocol)	80	tcp, udp
https (hypertext transfer protocol secure)	443	tcp, udp

Korisnički programi (i servisi) koriste najčešće broj porta veći od 1024.

6.3. Socket

Socket je krajnja tačka dvosmernog komunikacionog linka između dva programa koji se izvršavaju u mreži. Kako je socket vezan za broj porta TCP protokol može da identifikuje aplikaciju kojoj su namenjeni podaci koji se šalju.

Socket predstavlja uređeni par (IP adresa, port) jednog učesnika u komunikaciji. Za uspostavljanje veze između dva programa potrebna su dva socket-a (na svakoj komunikacionoj strani po jedan socket). Slika 6.3. ilustruje uspostavljenu vezu između dva programa korišćenjem socket-a `(192.168.0.15, 1234)` i socket-a `(192.168.0.21, 8765)`, respektivno na strani programa A i na strani programa B.



Slika 6.3. Veza dva programa pomoću socketa

Napomena je da je veza među programima, a ne između računara, što znači da

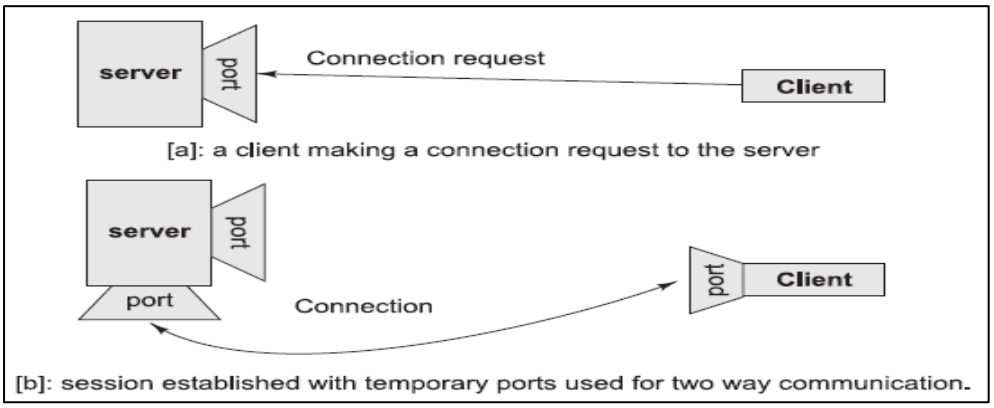
dva programa mogu komunicirati na istom računaru korišćenjem socket-a. Ovo je omogućeno različitim portovima koji imaju istu IP adresu.

Socket obezbeđuje interfejs za mrežno programiranje na transportnom nivou Internet steka. Mrežna komunikacija pomoću socket-a je vrlo slična obavljanju I/O operacija nad datotekama (socket hendler tretira se kao fajl hendler). Kao što se koriste u fajl I/O operacijama stream-ovi se koriste i na I/O operacijama baziranim na socket-ima.

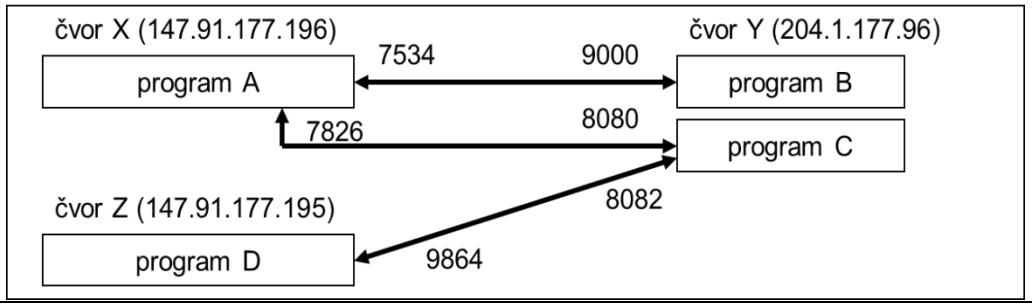
Komunikacija bazirana na socket-ima je nezavisna od programskog jezika koji se koristi za njenu realizaciju. Socket program napisan u Java jeziku može komunicirati sa socket programom pisanim u ne-Java (npr. u C++ jeziku).

Serverski program kreira socket koji je vezan za određeni port na kom se osluškuju zahtevi klijenata. Kada klijent zatraži konekciju sa serverom (slika. 6.4.a) server prihvata vezu i kreira socket na strani servera koji će nastaviti komunikaciju sa socketom prihvaćenog klijenta (slika 6.4.b) npr. u paralelnoj niti. Sada server ponovo može da ide u stanje osluškivanja klijenata na portu za osluškivanje.

Iz prethodnog sledi da je moguće u jednom programu implementirati više komunikacija korišćenjem različitih parova (IP,port).



Slika 6.4. Uspostavljanje dvosmerne komunikacije između klijenta i servera a) klijent šalje serveru zahtev za konekciju b)sesija je uspostavljena preko kreiranog scket-a na strani servera



Slika 6.5. Slučaj više uspostavljenih veza između programa

Na slici 6.5. prikazana je komunikacija između nekoliko programa: programa A na čvoru X, programa B i C na čvoru Y i programa D na čvoru Z. Na krajevima strelica dati su brojevi portova preko kojih ide komunikacija npr. na čvoru Z (IP, port)=(147.91.177.195, 9864).

Odgovarajući parovi u komunikaciji su kao što sledi:

- Program A na čvoru X komunicira
  - preko socket-a (IP,port)=(147.91.177.196, 7534) sa programom B na čvoru Y kod koga je socket (IP,port)=(204.1.177.96, 9000)
  - preko socket-a (IP,port)=(147.91.177.196, 7826) sa programom C na čvoru Y kod koga je socket (IP,port)=(204.1.177.96, 8080)
- Program D na čvoru Z komunicira
  - preko socket-a (IP,port)=(147.91.177.195, 9864) sa programom C na čvoru Y kod koga je socket (IP,port)=(204.1.177.96, 8082)

6.4. Java klase za mrežnu komunikaciju

Java klase za mrežnu komunikaciju nalaze se u paketu java.net. Ove klase omogućuju brz razvoj mrežnih aplikacija. Java klase koje se koriste za ovu komunikaciju podržavaju TCP (*Transport Control Protocol*) i UDP (*User Datagram Protocol*) protokole.

6.4.1. Klasa InetAddress

Klasa `InetAddress` u okviru paketa java.net. predstavlja klasu koja realizuje IP adresu. Objekat ove klase često se kreira pozivom statičke metode `getByName` klase `InetAddress` kojoj se prosleđuje parametar tipa String koji može biti:

- simbolički naziv čvora

```
InetAddress ipa=InetAddress.getByName("java.sun.com");
```

- bajt (oktet) zapis IP adrese čvora

```
InetAddress apa=InetAddress.getByName("147.91.177.196");
```

Metode `getByName` može da generiše izuzetak `UnknownHostException` ako ne može da odredi naziv računara.

Statička metoda `getLocalHost` generiše `InetAddress` objekat koji predstavlja adresu računara na kome se nalazi progam koji je poziva:

```
InetAddress c = InetAddress.getLocalHost();
```

Metoda `getAllByName` vraća niz objekata `InetAddress` ako jednan naziv sadrži više adresa. Ova metoda takođe može da generiše izuzetak

UnknownHostException.

Na slici 6.6. dat je primer koji koristi prethodne metode.

```
import java.net.*;
class InetAddressTest {
  public static void main(String args[])
                    throws UnknownHostException {
   InetAddress iadresa = InetAddress.getLocalHost();
   System.out.println(iadresa);
   iadresa = InetAddress.getByName("www.google.com");
   System.out.println(iadresa);
   InetAddress SW[]=InetAddress.getAllByName("www.nba.com");
   for (int i=0; i<SW.length; i++)
      System.out.println(SW[i]);
  }
}

Izlaz:
      default/206.148.209.138
      osborne.com/198.45.24.130
      www.nba.com/204.202.130.223
```

Slika 6.6. Upotreba metoda klase InetAddress

6.4.2. Klasa Socket

Za TCP komunikaciju koristi se klasa Socket. Potrebno je na obe strane u komunikaciji otvoriti konekciju. Otvaranje konekcije koja povezuje lokalni računar sa ciljnim računarom i brojem porta na ciljnom računaru može se realizovati na sledeće načine:

```
Socket s1=new Socket(addr,25); //addr je InetAddress objekat
Socket s2=new Socket("java.sun.com", 80);
```

Ove metode mogu da izazovu izuzetak tipa UnknownHostException ili IOException.

Za dobijanje adrese i porta koji određuju socket koriste se sledeće metode:

- `InetAddress getInetAddress();`

  vraća objekat klase InetAddress pridružen objektu klase Socket.
- `int getPort() ;`

  vraća port pridružen objektu klase Socket (port udaljenog priključka).
- `int getLocalPort();`

vraća lokalni port sa kojim je povezan pozivajući soket.

Nakon otvaranja konekcije preuzimaju se reference na stream objekte koji se koriste za slanje i primanje poruka. Inicijalizacija ulaznog stream-a preko koga će se prihvatati poruke od druge strane u komunikaciji kao i inicijalizacija izlaznog stream-a preko koga se šalju poruke drugoj strani u komunikaciji je kao što sledi:

```
// inicijalizacija ulaznog streama
BufferedReader ulaz =
  new BufferedReader(
      new InputStreamReader(
        nassocket.getInputStream()));
// inicijalizuj izlazni stream
PrintWriter izlaz =
  new PrintWriter(
      new BufferedWriter(
          new OutputStreamWriter(
              nassocket.getOutputStream())), true);
```

Metode `getInputStream` i `getOutputStream` klase `Socket` koriste se za generisanje ulaznog odnosno izlaznog srtream-a preko socketa. Sada se komunikacija sa programom sa kojim je uspostavljena konekcija vrši pozivanjem metoda klasa `BufferedReader` i `PrintWriter`:

```
izlaz.writeln("Koliko je sati ?");
String odgovor = ulaz.readLine(); //cita se odgovor
```

Zatvaranje konekcije vrši se zatvaranjem ulaznog i izlaznog stream-a i zatvaranje socket-a:

```
izlaz.close();
ulaz.close();
nassocket.close();
```

6.4.2.1. Klijent-server veza

U klijent/server arhitekturi klijent inicira komunikaciju i traži uslugu od servera. Komunikacija obuhvata parove pitanja (zahteva) klijenta i odgovora servera. Redosled događaja ide kao što sledi:

- pokrene se server
  - server osluškuje pozive
- pokrene se klijent
  - klijent šalje zahtev serveru
- server prihvata zahtev klijenta
- server šalje odgovor klijentu.

- klijent prihvata odgovor servera
- klijent prekida komunikaciju sa serverom.

6.4.2.2. Klijent

Klijent kreira socket koji ima par: IP adresa servera, port na serveru na kome se osluškuje poziv klijenta. Kod na slici 6.7. ilustruje inicijalizaciju (kreiranje socket-a, ulaznih i izlaznih stream-ova), komunikaciju (slanje zahteva servera i primanje odgovora servera) i prekid veze na strani klijenta.

```
InetAddress ipadresa= InetAddress.getByName("127.0.0.1");

Socket nassocket = new Socket(ipadresa, 8765);
BufferedReader ulaz =
  new BufferedReader(
      new InputStreamReader(
        nassocket.getInputStream()));
PrintWriter izlaz =
  new PrintWriter(
      new BufferedWriter(
          new OutputStreamWriter(
              nassocket.getOutputStream())), true);

// komunikacija klijenta sa serverom
out.println("zahtev klijenta");
String odgovorservera = in.readLine();

// klijent prekida konekciju sa serverom
ulaz.close();
izlaz.close();
nassocket.close();
```

Slika 6.7. Inicijalizacija klijenta

U prethodnom kodu server se nalazi na istom računaru (IP lokalna adresa 127.0.0.1) gde i klijentski program. Server će osluškivati klijenta na portu 8765.

6.4.3. Klasa ServerSocket

Za realizaciju servera koristi se klasa `ServerSocket`. Konstruktor ove klase ima formalne argumente:

- vrednost porta na kom će server osluškivati pozive klijenata
- dužina reda čekanja (nije obavezan parametar) koja ako se ne navede ima podrazumevanu vrednost 50.

IP adresa je adresa lokalnog računara na kom server osluškuje klijente tako da je dovoljan i samo parametar za port uz podrazumevanu dužinu reda čekanja.

```
ServerSocket ss = new ServerSocket(8765);
```

```
ServerSocket ss = new ServerSocket(8765,100);
```

Sada je potrebno osluškivati poziv klijenta na portu 8765 za šta se koristi sinhrona metoda `accept` kase `ServerSocket`. Kada se desi poziv klijenta ova metoda kreira socket na strani servera. Ovaj socket se koristi za komunikaciju sa socket-om klijenta koji je uspostavio vezu.

```
Socket s = ss.accept();
```

Ova metoda je sinhrona i čekaće sve dok ne stigne zahtev sa klijenta.

Kod na slici 6.8. ilustruje inicijalizaciju (kreiranje `ServerSocket` objekta koji će da osluškuje pozive na navedenom portu), komunikaciju (prihvatanje poziva osluškivanjem i kreiranjem socket-a na strani servera, kreiranje ulaznih i izlaznih stream-ova preko socket-a na strani servera, slanje odgovora) i zatvaranje konekcije prema klijentu.

```
ServerSocket server = new ServerSocket(8765);

// komunikacija sa inicijalizacijom socket-a na strani
// servera
Socket socketnaserverstrani = ss.accept();
BufferedReader ulaz =
  new BufferedReader(
      new InputStreamReader(
        socketnaserverstrani.getInputStream()));
PrintWriter izlaz =
  new PrintWriter(
      new BufferedWriter(
          new OutputStreamWriter(
             socketnaserverstrani.getOutputStream())),true);

// komunikacija
String zahtev = in.readLine();
out.println("slanje dogovora");

// server zatvara konkeciju prema klijentu
ulaz.close();
izlaz.close();
socketnaserverstrani.close();
```

Slika 6.8. Inicijalizacija servera

U datom primeru komunikacija klijenta i servera je trivijalna i svodi se na komunikaciju jednog klijenta i jednog servera.

6.4.3.1. Thread-ovana klijent-server arhitektura

Osnovna namena servera je da omogući opsluživanje više klijenata istovremeno. Osnovna ideja je da se svakom klijentskom socket-u koji poziva server dodeli socket koji će na serverskoj strani u paralelnoj niti da nastavi

komunikaciju sa klijentskim socket-om, pri čemu će se paralelno na strani servera ponovo otići u osluškivanje poziva novog klijenta.

Za n istovremenih klijenata postojaće n+2 programske niti. Za svakog klijenta po jedna, glavna nit za osluškivanje poziva i jedna za garbage collector. Na slici 6.9. dat je primer serverske strane koja za svaku vezu sa klijentom kreira nit koja obrađuje komunikaciju servera i klijenta.

```
// --- u glavnoj niti ---
...
ServerSocket server = new ServerSocket(brojportazaslusanje);
while (true) {
  Socket socketnaserverstrani = ss.accept();
  ServerThread nitzaklijenta =
               new ServerThread(socketnaserverstrani);
}
...
//
// --- u niti koja komunicira sa klijentom ---
class ServerThread extends Thread {
  // kod korisnika
  private Socket sock;
  public ServerThread(Socket sock){
     this.sock = sock;
     // kod korisnika
     start();
  }
  public void run() {
    // inicijalizacija
    // komunikacija
    // prekid veze
  }
}
```

Slika 6.9. Serverska strana koja koristi niti za komunikaciju sa klijentom

U datom primeru, kada se pojavi zahtev klijenta server će kreirati socket koji će biti prosleđen kao stvarni argument konstruktoru klase koja realizuje nit za komunikaciju sa klijentom (u primeru ServerThread). Sada se postavi da se referenca na privatni član tipa Socket odnosi na prosleđeni socket. Na kraju konstruktora pozove se metoda start klase Thread tako da se ova nit odmah i pokrene iz konstruktora.

U sledećem primeru:

- klijent
    - uspostavlja vezu sa serverom;

- zahteva od servera podatke o datumu i vremenu;
- čita odgovor servera i ispisuje ga na konzoli;
- završava komunikaciju;

- server
  - osluškuje klijente na datom portu;
  - prihvata vezu sa klijentom, kreira socket na strani servera i ažurira redni broj konekcije sa klijentom;
  - prosleđuje kreirani socket i redni broj konekcije objektu koji kao paralelna nit nastavlja da komunicira sa prihvaćenim klijentom;
  - ponovo osluškuje klijente na datom portu.
- nit na strani servera za komunikaciju sa klijentom
  - prihvata prosleđeni socket i redni broj konekcije od servera;
  - inicijalizuje ulazni i izlazni stream po prosleđenom socket-u;
  - pokreće se kao paralelna nit iz konstruktora koja komunicira sa klijentom;
  - prihvata zahtev klijenta i šalje odgovor klijentu;
  - zatvara svoje komunikacione resurse.

Pripadni kodovi za klijent, server, nit na strani servera za komunikaciju sa klijentom dati su na slikama 6.10, 6.11. i 6.12, respektivno. Na slici 6.13. prikazane su poruke koje ispisuje server, serverska nit i klijent.

```
import java.io.*;
import java.net.*;

public class Klijent{

  public static final int TCP_PORT = 9876;

  public static void main(String[] args) {

   try {
     InetAddress adresaservera =
                 InetAddress.getByName("127.0.0.1");
     Socket socketklijenta =
                 new Socket(adresaservera, TCP_PORT);
     BufferedReader ulaz=
     new BufferedReader(
     new
     InputStreamReader(socketklijenta.getInputStream()));
     PrintWriter izlaz =
      new PrintWriter(
        new BufferedWriter(
```

```
          new OutputStreamWriter(
            socketklijenta.getOutputStream())),true);
     System.out.println("[Klijent]: pokrenut...");
     System.out.println("[Klijent]: Datum i vreme?");
     izlaz.println("[Klijent]: Datum i vreme?");
     String odgovor="";
      while(!(odgovor=ulaz.readLine()).
                                  equals("KRAJ_PORUKE")){
         System.out.println(odgovor);
      }
      ulaz.close();
      izlaz.close();
      socketklijenta.close();
    }
    catch (UnknownHostException e1) {
      e1.printStackTrace();
    }
    catch (IOException e2) {
      e2.printStackTrace();
    }
    System.out.println("[Klijent]: kraj.");
  }
}
```

Slika 6.10. Klijentski kod TCP komunikacije

```
import java.io.*;
import java.net.*;
import java.util.*;

public class Server{

  public static final int TCP_SERVER_PORT = 9876;
  private static int redni_broj_konekcije = 0;

  public static void main(String[] args) {
    try {
      System.out.println("[Server]: pokrenut...");
      ServerSocket server =
                  new ServerSocket(TCP_SERVER_PORT);
      while (true) {
        System.out.println("[Server]: osluškuje...");
        Socket socketzaklijenta = server.accept();
        System.out.println("[Server]: dodeljen socket za "+
                (++redni_broj_konekcije)+". klijenta");
        ServerskaNit serverska_nit_za_komunikaciju =
        new ServerskaNit(socketzaklijenta,
                         redni_broj_konekcije);
```

```
        System.out.println("[Server]: aktivirana nit "+
                           "za komunikaciju sa klijentom "+
                           redni_broj_konekcije+".");
      }
    }
    catch (Exception ex) {
       ex.printStackTrace();
    }
  }
}
```

Slika 6.11. Serverski kod TCP komunikacije

```
import java.util.*;
import java.io.*;
import java.net.*;

public class ServerskaNit extends Thread {

  private Socket socket;
  private int redni_broj_konekcije;
  private BufferedReader ulaz;
  private PrintWriter izlaz;

  public ServerskaNit (Socket sock,
                       int redni_broj_konekcije){
    socket = sock;
    this.redni_broj_konekcije = redni_broj_konekcije;
    try {
      ulaz = new BufferedReader(
                new InputStreamReader(
                      socket.getInputStream()));
            izlaz = new PrintWriter(
                    new BufferedWriter(
                      new OutputStreamWriter(
                        socket.getOutputStream())),true);
      } catch (Exception ex) {
            ex.printStackTrace();
      }
      start();
 }

 public void run() {
    try {
      String zahtev = ulaz.readLine();
      System.out.println("[Serverska nit][Klijent broj "+
                         redni_broj_konekcije+"]: "+zahtev);
      Date datumivreme = new Date();
      izlaz.println(
```

```
            "[Serverska nit]: Vas redni broj konekcije je "+
            redni_broj_konekcije+"\n"+
            "[Serverska nit]: datum i vreme: "+
             datumivreme+"\n"+
             "KRAJ_PORUKE");
      ulaz.close();
      izlaz.close();
      socket.close();
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
    System.out.println("[Serverska nit]: kraj. ");
  }
}
```

Slika 6.12. Nit na strani servera za TCP komunikaciju sa klijentom

Server	**Serverska nit**	**Klijent**
[Server]: pokrenut...		
[Server]: osluškuje...		
		[Klijent]: pokrenut
[Server]: dodeljen socket za 1. klijenta		
[Server]: aktivirana nit za komunikaciju sa klijentom 1.		[Klijent]: Datum i vreme?
[Server]: osluškuje...	[Serverska nit][Klijent broj 1]: [Klijent]: Datum i vreme?	
		[Serverska nit]: Vas redni broj konekcije je 1 [Serverska nit]: datum i vreme: Sat Jan 26 22:13:28 CEST 2019
	(salje KRAJ_PORUKE) [Serverska nit]: kraj.	
		[Klijent]: kraj.

Slika 6.13. Poruke koje ispisuje server, serverska nit i klijent

Da bi server naizmenično davao svakih n sekundi poruku da radi (obično bude kao deo zadatka na ispitu) ispisujući dve poruke naizmenično sa malom pauzom npr. server radi... i SERVER RADI... potrebno je da se modifikuje kod za server jer je metoda accept klase ServerSocket sinhrona metoda (slika 6.14).

```
//server koji jos ispisuje da radi
import java.io.*;
import java.net.*;
import java.util.*;

public class Server extends Thread{

  public static final int TCP_SERVER_PORT = 9000;
  private static int redni_broj_konekcije = 0;
  private static final int pauza = 5000;

  public void run(){
      while(true){
           try {
               System.out.println("server radi...");
               sleep(pauza);
               System.out.println("SERVER RADI...");
               sleep(pauza);
           } catch (InterruptedException ex) {
               System.out.println(ex.getMessage());
           }
      }
  }

  public static void main(String[] args) {
    try {
      Server s = new Server();
      s.start();
      System.out.println("[Server]: pokrenut...");
      ServerSocket server =
                  new ServerSocket(TCP_SERVER_PORT);
      while (true) {
        System.out.println("[Server]: osluškuje...");
        Socket socketzaklijenta = server.accept();
        System.out.println("[Server]: dodeljen socket za "+
                (++redni_broj_konekcije)+". klijenta");
        ServerskaNit serverska_nit_za_komunikaciju =
        new ServerskaNit(socketzaklijenta,
                         redni_broj_konekcije);
        System.out.println("[Server]: aktivirana nit "+
                            "za komunikaciju sa klijentom "+
                            redni_broj_konekcije+".");
```

```
      }
    }
    catch (Exception ex) {
       ex.printStackTrace();
    }
  }
}
```

Slika 6.14. Ispisivanje dodatnih poruka na serveru svakih 5 sekundi

Mogući izlaz (konzola servera) koji se ispisuje na serverskoj strani je kao što sledi:

```
[Server]: pokrenut...
server radi...
[Server]: osluškuje...
SERVER RADI...
[Server]: dodeljen socket za 1. klijenta
[Server]: aktivirana nit za komunikaciju sa klijentom 1.
[Server]: osluškuje...
[Serverska nit][Klijent broj 1]: [Klijent]: Datum i
vreme?
[Serverska nit]: kraj.
server radi...
SERVER RADI...
server radi...
```

U primeru se poruke između servera i klijenta razmenjuju kao stringovi gde je graničnik oznaka za kraj reda. Metoda `readLine` je sinhrona i čeka dok se na ulazu ne pojavi znak za novi red.

U slučaju da se zna broj znakova u datoj poruci u komunikaciji se implementira čitanje upravo toliko znakova koliko stiže sa ulaza.

```
ulaz.read(buffer, 0, brojznakova);
```

U ovakvoj komunikaciji protokoli nose informaciju o broju znakova koji se šalju u jednoj poruci u zaglavlju poruke tako da prijemna strana dobija podatak koliko se znakova šalje. Moguće je realizovati poruke fiksne dužine ili napraviti kombinaciju prethodna dva slučaja (kada se koriste različiti tipovi poruka).

Sledi primer koji je lepa osnova za seminarske radove. Kreira se serverski program koji obavlja neke matematičke operacije. Server parsira klijentov zahtev i šalje odgovarajući odgovor. Naravno, klijenti mogu biti realizovani i u drugim programskim jezicima, ali uz poštovanje (dogovorenog protokola) formata poruka koje se razmenjuju sa serverom.

Program je realizovan kreiranjem: osnovnog interfejsa za servise koje pruža server, implementacije osnovnog interfejsa, klijentskog i serverskog programa.

Server čeka da mu klijent prosledi zahtev koji mora imati format

operacija:prvi_operand:drugi_operand. Trivijalnom parserom analizira se zahtev klijenta, poziva odgovarajuća metoda klase koja implementira osnovni interfejs koja odgovara traženoj operaciji (servisu). Ova metoda vraća rezultat koji se prosleđuje klijentu.

Na slikama 6.15, 6.16, 6.17. i 6.18. dati su kodovi: interfejsa MathService, klase PlainMathService koja implementira prethodno navedeni interfejs, klijentske strane i serverske strane, respektivno.

```
public interface MathService {
    public double add(double firstValue, double secondValue);
    public double sub(double firstValue, double secondValue);
    public double div(double firstValue, double secondValue);
    public double mul(double firstValue, double secondValue);
}
```

Slika 6.15. Interfejs MathService

```
public class PlainMathService implements MathService {
  public double add(double firstValue, double secondValue) {
        return firstValue + secondValue;
    }
  public double sub(double firstValue, double secondValue) {
        return firstValue - secondValue;
    }
  public double mul(double firstValue, double secondValue) {
        return firstValue * secondValue;
    }
  public double div(double firstValue, double secondValue) {
    if (secondValue != 0) {
       return firstValue / secondValue;
    }
    return Double.MAX_VALUE;//ili da se resi preko izuzetka
  }
}
```

Slika 6.16. Klasa PlainMathService koja implementira interfejs MathService

```
import java.io.*;
import java.net.Socket;

public class MathClient {

    public static void main(String[] args) {
        String hostname = "localhost";
        int port = 10000;
        if (args.length != 2) {
           System.out.println("Podrazumevano podesavanje.");
        }
        else {
```

```
            hostname = args[0];
            port = Integer.parseInt(args[1]);
        }
        try {
          // kreiranje socket-a
          Socket socket = new Socket(hostname, port);
          // trivijalna operacija 12+21
          BufferedWriter writer = new BufferedWriter(
          new OutputStreamWriter(socket.getOutputStream()));
          writer.write("+:12:21");
          writer.newLine();
          writer.flush();
          // čitanje rezultata sa servera
          BufferedReader reader = new BufferedReader(
            new InputStreamReader(socket.getInputStream()));
          System.out.println(reader.readLine());
          reader.close();
          writer.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Slika 6.17. Klijentski program

```
import java.io.*;
import java.net.*;

public class MathServer {

    protected MathService mathService;
    protected Socket socket;

    public void setMathService(MathService mathService) {
        this.mathService = mathService;
    }
    public void setSocket(Socket socket) {
        this.socket = socket;
    }
    public void execute() {
        try {
          BufferedReader reader = new BufferedReader(
            new InputStreamReader(socket.getInputStream()));
          // citanje poruke kllijenta,parsiranje,izvršavanje
          String line = reader.readLine();
          double result = parseExecution(line);
          // slanje rezultata klijentu
```

```
      BufferedWriter writer = new BufferedWriter(
      new OutputStreamWriter(socket.getOutputStream()));
      writer.write("" + result);
      writer.newLine();
      writer.flush();
      // zatvaranje stream-a
      reader.close();
      writer.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
// predefinisani protokol za matematičku operaciju je:
// operator:first_value:second_value
protected double parseExecution(String line)
                throws IllegalArgumentException {
  double result = Double.MAX_VALUE;
  String[] elements = line.split(":");
  if (elements.length != 3) {
    throw new IllegalArgumentException("Parsiranje!!!");
  }
  double firstValue = 0;
  double secondValue = 0;
  try {
      firstValue = Double.parseDouble(elements[1]);
      secondValue = Double.parseDouble(elements[2]);
   }
   catch (Exception e) {
     throw new IllegalArgumentException("Argumenti!!!");
   }
   switch (elements[0].charAt(0)) {
     case '+':
      result = mathService.add(firstValue, secondValue);
      break;
     case '-':
      result = mathService.sub(firstValue, secondValue);
      break;
     case '*':
      result = mathService.mul(firstValue, secondValue);
      break;
     case '/':
      result = mathService.div(firstValue, secondValue);
      break;
     default:
```

```
          throw new IllegalArgumentException("Operacija!!!");
       }
       return result;
    }

    public static void main(String[] args)
                        throws Exception {
        int port = 10000;
        if (args.length == 1) {
            try {
                port = Integer.parseInt(args[0]);
            } catch (Exception e) {
            }
        }
        System.out.println("Server pokrenut ...");
        //kreiranje SserverSocket-a
        ServerSocket serverSocket = new ServerSocket(port);
        // cekanje na poziv klijenta
        Socket socket = serverSocket.accept();
        // start matematickog servera koji
        // komunicira sa klijentom
        MathServer mathServer = new MathServer();
        mathServer.setMathService(new PlainMathService());
        mathServer.setSocket(socket);
        mathServer.execute();
        System.out.println("Server zavrsio rad.");
    }
}
```

Slika 6.18. Serverski program

Ispisi na strani servera i klijenta su kao što sledi:

Serverska strana	Klijentska strana
Server pokrenut ...	
	Podrazumevano podesavanje.
	33.0
Server zavrsio rad	

Izmenite kod na serverskoj strani tako da se dobije klijent-server thread-ovana arhitektura !

6.4.4. Klasa FileDialog i slanje fajla sa servera

U sledećem primeru se odabrani fajl šalje klijentima na njihov zahtev. Na serverskoj strani se bira fajl koji će biti postavljen za preuzimanje od strane klijenata.

Klasa `TrivialFileServer` realizovana je u `Swing` GUI tehnologiji i implementira interfejs `ActionListener` (slika 6.19). Kontejner c sadrži programsko dugme `odaberifajl` i tekstualnu oblast `jta` koji su raspoređeni kao matrica 2x1 (`GridLayout(2,1)`) (slika 6.20). Reakcija na događaj klik na programsko dugme `odaberifajl` aktivira dijalog koji omogućuje biranje fajla koji će biti postavljen za preuzimanje od strane klijenata. Kada se odabere dati fajl, onda se iz kontejnera uklanja programsko dugme `odaberifajl`, vrši promena raspoređivača komponenti (`GridLayout(1,1)`) tako da se sada prikazuje samo oblast za tekst i poziva metoda `revalidate()` koja će osvežiti prikaz kontejnera.

Metoda `Ispisi` omogućuje da se u oblasti za tekst ispisuje formatirano vreme (korišćenjem `SimpleDateFormat`) i prosleđeni string. Metoda `Ok` vraća referencu na naziv odabranog fajla. Sve dok fajl ne bude odabran ne ide se na osluškivanje klijenata.

U `main` metodi za baratanje fajlom koristi se par `FileInputStream` (identifikaotr `fis`) i `BufferedInputStream` (identifikator `bis`). `Main` metoda može da baci izuzetak tipa `IOException`. Ideja je da se u niz bajtova, bafer (identifikator `mybytearray`), smesti sadržaj odabranog fajla korišćenjem metode `read` klase `BufferedInputStream` čiji su parametri odredište, indeks od koga počinje smeštanje bajtova i dužina niza (u bajtovima) koji se smešta.

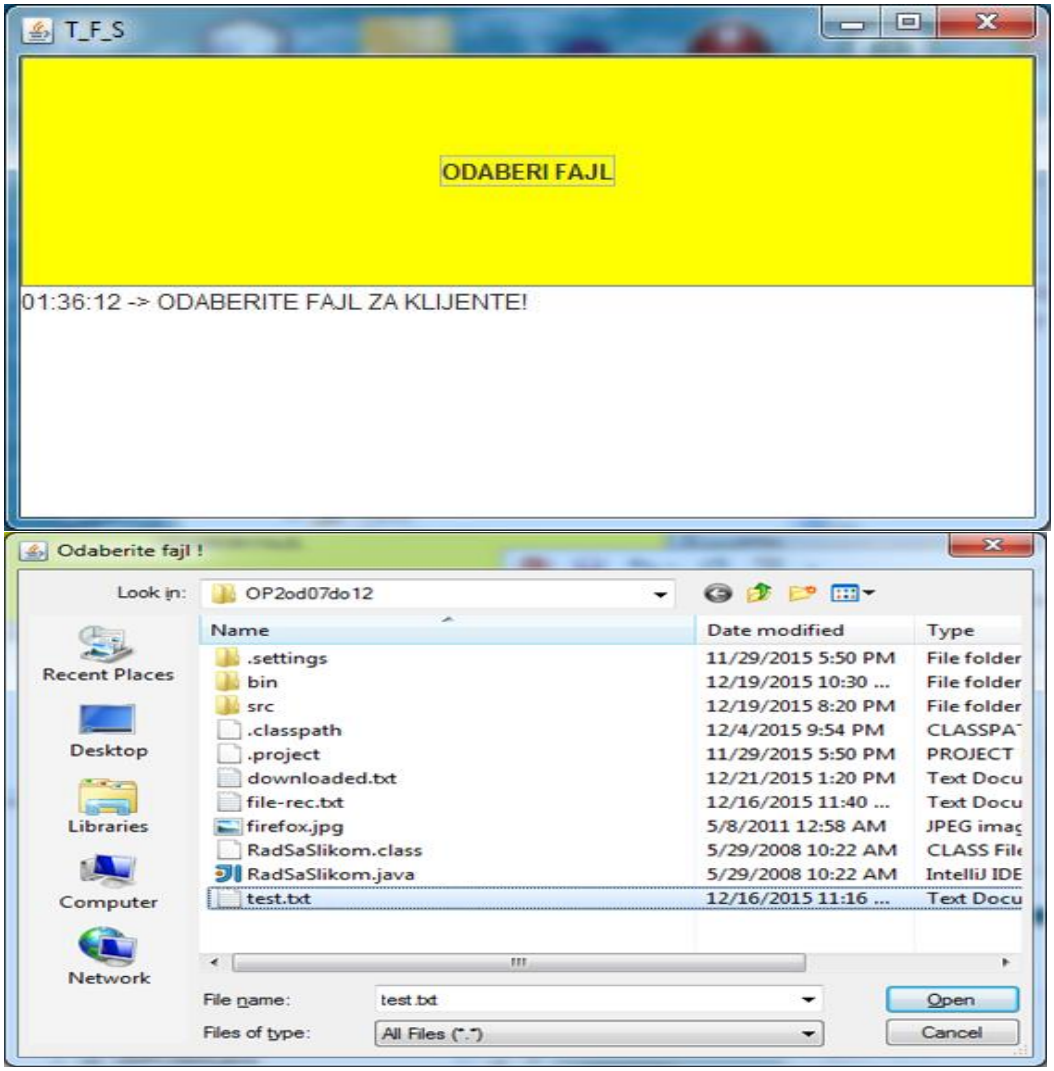
Pre petlje za opsluživanje klijenata kreira se server soket (`ServerSocket`). U petlji za opsluživanje klijenata:

- u oblast za tekst ispisuje se informacija da se čeka na poziv klijenta sa pripadnim vremenom;
- čeka se na prihvatanje klijenta;
- za prihvaćeni zahtev se definiše soket na strani servera (identifikator `sock`) koji je vezan sa soketom klijenta;
- referenca na `OutputStream` (identifikator `os`) se veže za `OutputStream` soketa `sock`;
- podaci o fajlu koji se šalje i njegovoj veličini se ispisuju u oblast za tekst;
- pozivom metode `os.write` šalju se bajtovi iz bafera u soketov izlazni tok;
- poziva se metoda `os.flush()` za pražnjenje bafera izlaznog toka;
- u oblast za tekst ispisuje se vreme i poruka da su podaci poslani;
- konačno, oslobađaju se resursi koji su bili zauzeti.

```
import java.util.*;
import java.io.*;
import java.net.*;
import java.text.SimpleDateFormat;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TrivialFileServer extends JFrame implements
ActionListener {
  public final static int SOCKET_PORT = 12345;
  private JButton odaberifajl;
  private FileDialog filedialog;
  private volatile String strfajl;
  private JTextArea jta;
  public TrivialFileServer() {
    setTitle("T_F_S");     setSize(500, 300);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Container c = this.getContentPane();
    odaberifajl = new JButton("ODABERI FAJL");
    odaberifajl.setBackground(Color.YELLOW);
    odaberifajl.addActionListener(this);
    filedialog = new FileDialog(this, "Odaberite fajl !");
    strfajl= null;
    jta = new JTextArea(30,20);
    Ispisi("ODABERITE FAJL ZA KLIJENTE!");
    c.setLayout(new GridLayout(2,1));
    c.add(odaberifajl);     c.add(jta);
    setVisible(true);
  }
  public String Ok(){ return strfajl; }
  public void Ispisi(String str){
    Date dt = new Date();
    SimpleDateFormat sdf = new SimpleDateFormat("hh:mm:ss");
    String time1 = sdf.format(dt);
    jta.insert(time1+" -> " +str+"\n", 0);
  }
  public void actionPerformed(ActionEvent ae) {
    if(ae.getActionCommand().equals("ODABERI FAJL")){
      filedialog.setVisible(true);
      strfajl = filedialog.getFile();
      if(strfajl!= null){
        Ispisi("ODABRAN: "+strfajl);
        this.getContentPane().remove(odaberifajl);
        this.getContentPane().setLayout(new GridLayout(1,1));
        revalidate();
      }
```

```
    }
  }
  public static void main(String[] args) throws IOException {
    TrivialFileServer tfs = new TrivialFileServer();
    FileInputStream fis = null;
    BufferedInputStream bis = null;
    OutputStream os = null;
    ServerSocket servsock = null;
    Socket sock = null;
    while(tfs.Ok()== null);
    try {
      servsock = new ServerSocket(SOCKET_PORT);
      File myFile = new File(tfs.Ok());
      byte[] mybytearray = new byte[(int) myFile.length()];
      fis = new FileInputStream(myFile);
      bis = new BufferedInputStream(fis);
      bis.read(mybytearray, 0, mybytearray.length);
      while (true) {
        tfs.Ispisi("CEKAM POZIV KLIJENTA ...");
        try {
          sock = servsock.accept();
          tfs.Ispisi("Prihvacen poziv: " + sock);
          // send file
          os = sock.getOutputStream();
          tfs.Ispisi("Saljem " + tfs.Ok() + " (" +
                     mybytearray.length + " bytes)");
          os.write(mybytearray, 0, mybytearray.length);
          os.flush();
          tfs.Ispisi("Poslato.");
        } finally {
            if (fis != null) fis.close();
            if (bis != null)  bis.close();
            if (os != null)  os.close();
            if (sock != null)sock.close();
        }
      }
    } finally {
        if (servsock != null) servsock.close();
    }
  }
}
```

Slika 6.19. Slanje fajla sa servera: kod na strani servera



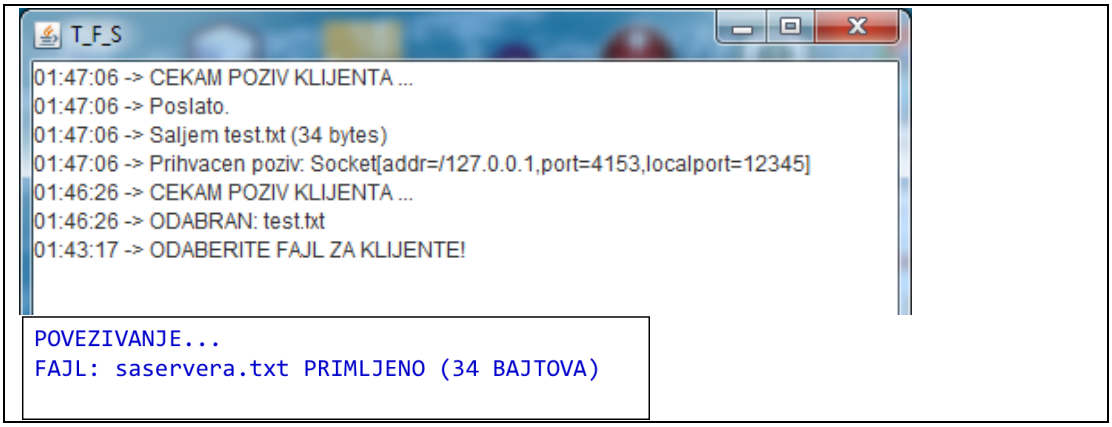
Slika 6.20. Početni igled serverske aplikacije (gore), izgled fajl dijaloga (dole)

Na klijentskoj strani:

- definiše se naziv pod kojim će biti snimljen fajl koji se dobija sa servera;
- koristi se par `FileOutputStream` (identifikato `fos`), `BufferedOutputStream` (identifikator `bos`) za snimanje fajla na stani klijenta;
- kreira se potrebni soket;
- ispisuje se informacija u konzoli klijenta da se ide na povezivanje (sa serverom);
- u bafer date veličine (uradite dinamički) biće smeštani bajtovi koji se dobijaju preko ulaznog toka soketa (metoda `read` klase `InputStream`);
- metoda `read` klase `InputStream` sa parametrima: odredište, indeks od kog se smeštaja niz bajtova i veličina niza, vraća koliko je bajtova pročitano (smešteno) pri pozivu ove metode, a onda se taj broj smesti u indeks `current`;

- pozivi metode read ide sukcesivno dok ima pristiglih bajtova, pri čemu se ti delovi sukcesivno snimaju od sledećeg slobodnog indeksa u baferu za šta se koristi indeks `current` koji se sukcesivno i ažurira;
- kada su svi bajtovi fajla sa servera učitani u bafer, onda se poziva metoda `write` klase `BufferedOutputStream` čime se podaci iz bafera smeštaju u izlazni tok (`bos`);
- poziva se metoda `flush()` klase `BufferedOutputStream` čime se svi podaci iz izlaznog toga smeste u datoteku "`saservera.txt`";
- na konzoli klijenta ispisuje se u koji je fajl sve smešteno i kolika je njegova veličina (slika 6.21);
- na kraju se oslobađaju zauzeti resursi.

Videti sliku 6.22.



Slika 6.21. Slanje fajla sa servera: ispis na serveru i klijentu

```
import java.io.*; import java.net.*;
public class TrivialFileClient {
  public final static int SOCKET_PORT = 12345;
  public final static String SERVER = "127.0.0.1";
  public final static String FILE_TO_RECEIVED = "saservera.txt";
  public final static int FILE_SIZE = 1000000; //uradite bez ovoga
  public static void main(String[] args) throws IOException {
    int bytesRead;
    int current = 0;
    FileOutputStream fos = null;
    BufferedOutputStream bos = null;
    Socket sock = null;
    try {
      sock = new Socket(SERVER, SOCKET_PORT);
      System.out.println("POVEZIVANJE...");
      // receive file
      byte[] mybytearray = new byte[FILE_SIZE];
      InputStream is = sock.getInputStream();
```

```
      fos = new FileOutputStream(FILE_TO_RECEIVED);
      bos = new BufferedOutputStream(fos);
      bytesRead = is.read(mybytearray, 0, mybytearray.length);
      current = bytesRead;
      do {
        bytesRead = is.read(mybytearray, current,
                                    (mybytearray.length -current));
        if (bytesRead > 0) current += bytesRead;
      } while (bytesRead > -1);
      bos.write(mybytearray, 0, current);
      bos.flush();
      System.out.println("FAJL: " + FILE_TO_RECEIVED +
                         " PRIMLJENO (" + current + " BAJTOVA)");
    } finally {
        if (fos != null)  fos.close();
        if (bos != null)  bos.close(); if (is != null) is.close();
        if (sock != null) sock.close();
    }
  }
}
```

Slika 6.22. Slanje fajla sa servera: kod klijenta

Probajte da unapredite kod servera i kod klijenta.

6.4.5. Klase DatagramSocket i DatagramPacket

U prethodnim primerima korišćeni su TCP socket-i. Ranije je pomenuto da TCP garantuje isporuku paketa i čuvanje njihovog redosleda kojim pristižu na odredište. Ova osobina ima i svoju cenu u smislu efikasnosti prenosa.

Kada navedene osobine nisu potrebne može se koristiti UDP protokol. Ovaj protokol prenosi datagram pakete. Datagram paketi se koriste za realizaciju beskonekcionih (*connection-less*) paketa. Svaka poruka se prenosi od izvora do odredišta na osnovu podataka sadržanih unutar tog paketa. Svaki paket mora imati adresu odredišta i svaki paket može biti preusmeren drugačije, a može stići i u bilo kom redosledu. Dostava ovakvih paketa nije garantovana.

Na slici 6.23. dat je format datagram paketa koji sadrži poruku, dužinu poruke, odredišni host i odredišni port.

Message	Length	Host	Server Port

Slika 6.23. Format datagram paketa

Java klase za podršku komunikacije korišćenjem datagrama su:

- `DatagramPacket(byte[] buf, int length,`
  `InetAddress address, int port);`

- Navedeni konstruktor koristi se za kreiranje datagram paketa za slanje paketa date dužine na odgovarajuću IP adresu i dati broj porta. Poruka se smešta u prvi argument.

  Ostali konstruktori su:
  - `DatagramPacket (byte[] buf, int length)`
  - `DatagramPacket (byte[] buf, int offset, int length)`
    - zadaje pomak u baferu gde će se smeštati podaci.
  - `DatagramPacket (byte[] buf, int offset, int length, InetAddress ipAdresa, int port)`
- Značajne metode klase `DatagramPacket` su:
  - `byte[] getData()`
    - vraća bafer podataka
  - `int getLength()`
    - vraća dužinu podataka koji se šalju ili dužinu prispelih podataka.
  - `void setData(byte[] buf)`
    - postavlja podatke za tekući paket.
  - `void setLength(int length)`
    - postavlja dužinu tekućeg paketa.
  - `InetAddress getAddress()`
    - vraća odredišnu adresu.
  - `int getPort()`
    - vraća broj priključka odredišta.

- `DatagramSocket(int port);`
  - Navedeni konstruktor kreira datagram socket koji koristi dati port
  - Ključne metode klase `DatagramSocket` su:
  - `void send(DatagramPacket dp)`
    - šalje datagram paket na socket.
  - `void receive(DatagramPacket dp)`
    - prihvata datagram paket sa socket-a.
  - `InetAddress getInetAddress()`
    - vraća odredišnu adresu.
  - `InetAddress getLocalAddress()`
    - vraća lokalnu adresu.
  - `int getPort()`
    - vraća broj priključka odredišta.
  - `int getLocalPort()`
    - vraća broj lokalnog priključka.

Sledi komunikacioni program koji koristi slanje datagrama. Jednostavan UDP

eho serverski program čeka zahtev klijenta. Po prijemu zahteva server klijentu vraća prihvaćenu poruku (datagram). Videti sliku 6.24. Kod klijenta je dat na slici 6.25.

```
import java.net.*;
import java.io.*;

public class UDPServer {
    public static void main(String args[]) {
        DatagramSocket aSocket = null;
        if (args.length < 1) {
            System.out.println("Usage:
                              java UDPServer <Port Number>");
            System.exit(1);
        }
        try {
           int socket_no =
              Integer.valueOf(args[0]).intValue();
            aSocket = new DatagramSocket(socket_no);
            byte[] buffer = new byte[1000];
            while (true) {
                System.out.println("cekam na klijenta ...");
                DatagramPacket request =
                new DatagramPacket(buffer, buffer.length);
                aSocket.receive(request);
                DatagramPacket reply =
                   new DatagramPacket(
                        request.getData(),
                        request.getLength(),
                        request.getAddress(),
                        request.getPort());
                System.out.println("saljem klijentu ...");
                aSocket.send(reply);
                System.out.println("POSLAO !");
            }
        } catch (SocketException e) {
            System.out.println(
                    "Socket: " + e.getMessage());
        } catch (IOException e) {
            System.out.println(
                    "IO: " + e.getMessage());
        } finally {if (aSocket != null) { aSocket.close();       }
        }
    }
}
```

Slika 6.24. Serverska strana za UDP komunikaciju

```
import java.net.*;
import java.io.*;

public class UDPClient {

    public static void main(String args[]) {
        // args give message contents and server hostname
        DatagramSocket aSocket = null;
        if (args.length < 3) {
            System.out.println(
              "Usage: java UDPClient <message > <Host name > <Port
number >");
            System.exit(1);
        }
        try {
            aSocket = new DatagramSocket();
            InetAddress aHost =
                            InetAddress.getByName(args[1]);
            int serverPort =
                        Integer.valueOf(args[2]).intValue();
            byte[] m = args[0].getBytes();
            DatagramPacket request =
                    new DatagramPacket(m,
                                       args[0].length(),
                                       aHost,
                                       serverPort);
            aSocket.send(request);
            System.out.println("poslao !");
            byte[] buffer = new byte[1000];
            DatagramPacket reply = new
                      DatagramPacket(buffer, buffer.length);
            aSocket.receive(reply);
            System.out.println("Server vraca: " +
                               new String(reply.getData()));
        }
        catch (SocketException e) {
            System.out.println("Socket: " + e.getMessage());
        }
        catch (IOException e) {
            System.out.println("IO: " + e.getMessage());
        }
        finally {
            if (aSocket != null) {
                aSocket.close();
            }
```

```
        }
    }
}
```

Slika 6.25. Klijentska strana za UDP komunikaciju

Izlaz:

serverska strana	klijentska strana
cekam na klijenta ...	
	poslao !
saljem klijentu ...	
POSLAO ! cekam na klijenta ...	Server vraca: Pozdrav od klijenta !

Proširite kod klijenta i kod servera novim funkcionalnostima !

6.4.6. Klasa URL

URL (*Uniform Resource Locator*) predstavlja jedinstven naziv dodeljen svakom resursu na Internetu. Java ima podršku za URL adresni pristup resursima Interneta.

Primer URL adrese:

```
http://www.myedu.net
http://www.myedu.net:80/index.htm.
```

Delovi URL adrese su kao što sledi:

- protokol
  - odvojen je dvotačkom od ostaka URL adrese.
- naziv računara ili IP adresa računara
  - levi graničnik je dvostruka kosa crta, a desni graničnik je kosa crta ili dvotačka ako se navodi port.
- port
  - neobavezan parametar. Levi graničnik je dvotačka, a desni graničnik je kosa crta.

    Ako je protokol HTTP, onda je podrazumevano port 80.
- navigacija do fajla
  - u datom primeru index.htm

URL klasa može izazvati izuzetak `MalformedURLException`. Konstruktori URL klase su kao što sledi:

- `URL (String urlString)`
- `URL(String protokol, String racunar, int port,`

```
      String putanja)
```

- `URL (String protokol, String racunar, String putanja)`

Na slici 6.26. je dat primer koji ispisuje svojstva stranice Osborne (`http://www.osborne/download`).

```
import java.net.*;

class URLDemo {

  public static void main(String args[])
                    throws MalformedURLException {

    URL hp = new URL("http://www.osborne/download");
    System.out.println("Protokol:\t" + hp.getProtocol());
    System.out.println("Priključak:\t" + hp.getPort());
    System.out.println("Računar:\t" + hp.getHost());
    System.out.println("Datoteka:\t" + hp.getFile());
    System.out.println("URL:\t" + hp.toExternalForm());
  }
}
```

Slika 6.26. Ispisivanja svojstva VEB stranice

```
Izlaz:
      Protokol:   http
      Priključak: -1
      Računar:    www.osborne
      Datoteka:   /download
      URL:  http://www.osborne/download
```

6.4.7. Klasa URLConnection

Klasa `URLConnection` (u `java.net` paketu) koristi se za pristup sadržaju udaljenih resursa. Ovim se mogu proveriti svojstva udaljenih objekata.

Sledi primer (slika 2.27) koji koristi metodu `openConnection` klase `URLConnection` da bi se kreirao objekat `URLConnection`. Program uspostavlja `HTTP` vezu sa lokacijom `http://www.w3.org/`, lista vrednosti zaglavlja i učitava sadržaj.

```
import java.net.*;
import java.io.*;
import java.util.Date;

class URLConnectionPrimer
{
  public static void main(String args[]) throws Exception {
```

```
    int c;
    URL hp = new URL("http://www.w3.org/");
    URLConnection hpCon = hp.openConnection();

    System.out.println("Datum: "+new Date(hpCon.getDate()));
    System.out.println("Vrsta sadržaja: " +
                        hpCon.getContentType());
    System.out.println("Rok trajanja: " +
                        hpCon.getExpiration());
    System.out.println("Vreme poslednje izmene: " +
                        new Date(hpCon.getLastModified()));
    int len = hpCon.getContentLength();
    System.out.println("Dužina sadržaja: " + len);
    if (len > 0) {
      System.out.println("Sadržaj:");
      InputStream input = hpCon.getInputStream();
      int i = len;
      while (((c = input.read()) != -1) && (--i > 0)) {
        System.out.print((char) c);
      }
      input.close();
    }
    else {
       System.out.println("Nema dostupnih podataka"); }
  }
}
```

Slika 6.27. Upotreba klase URLConnection

```
Izlaz:
Datum: Mon Oct 08 16:48:51 GMT+01:00 2012
Vrsta sadržaja: text/html; charset=utf-8
Rok trajanja: 1349711931000
Vreme poslednje izmene: Mon Oct 08 16:27:25 GMT+01:00 2012
Dužina sadržaja: 34169
Sadržaj:
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
lang="en">
<!-- Generated from data/head-home.php, ../../smarty/{head.tpl} --
>
<head>
<title>World Wide Web Consortium (W3C)</title>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
...
</body>
</html>
```

Za sledeći primer potrebna je klasa za URL kodiranje ne-ASCII karaktera (slika 6.28). Koristi se metoda encode klase `URLEncoder` da bi se string kodovao u dati format ("`UTF-8`"). Koristi se klasa `StringBuilder` (promenljiva tj. *mutable* klasa) i njena metoda append za konkatenaciju.

```
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
public class QueryStringFormatter {
   private String queryEngine;
   private StringBuilder query = new StringBuilder();
   public QueryStringFormatter(String queryEngine) {
      this.queryEngine = queryEngine;
   }
   public String getEngine() {
       return this.queryEngine;
   }
   public void addQuery(String queryKey, String queryValue)
          throws Exception {
     query.append(queryKey + "="
             + URLEncoder.encode(queryValue, "UTF-8") + "&");
   }
   public String getQueryString() {
       return "?" + query.toString();
   }
}
```

Slika 6.28. Slanje upita pretraživaču

Klasa `URLConnection` poseduje `getInputStream` i `getOutputStream` metode slične metodama `getInputStream` i `getOutputStream` klase `Socket`.

Sledeći primer (slika 6.29) demostrira korišćenje klase `URLConnetion` i `URLEncoder` za slanje upita na pretraživač Yahoo. Program kreira kodirani upit koji će koristiti web aplikacija, a onda šalje i prima podatke u sprezi sa Yahoo pretraživačem.

```
import java.io.*;
import java.net.*;
public class UpitZaYahoo {
  private String searchEngine;
```

```
  public UpitZaYahoo(String searchEngine) {
     this.searchEngine = searchEngine;}
  public void doSearch(String queryString) {
    try {
      // otvaranje url konekcije
      URL url = new URL(searchEngine);
      URLConnection connection = url.openConnection();
      connection.setDoOutput(true);
      // slanje upita pretrazivacu
      PrintStream ps = new
          PrintStream(connection.getOutputStream());
      ps.println(queryString);
      ps.close();
      // citanje i ispisivanje rezultata
      DataInputStream input =
        new DataInputStream(connection.getInputStream());
      BufferedReader lines =
        new BufferedReader(new InputStreamReader(input, "UTF-8"));
      String inputLine = null;
      while ((inputLine = lines.readLine()) != null) {
                System.out.println(inputLine);
      }
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
  public static void main(String[] args) throws Exception {
    QueryStringFormatter formatter =
      new QueryStringFormatter("http://search.yahoo.com/search");
    formatter.addQuery("newwindow", "1");
    formatter.addQuery("q", "Bruce Lee & Ip Man");
    // pretrazivanje pomocu yahoo-a
    UpitZaYahoo search = new UpitZaYahoo(formatter.getEngine());
    search.doSearch(formatter.getQueryString());
  }
}
```

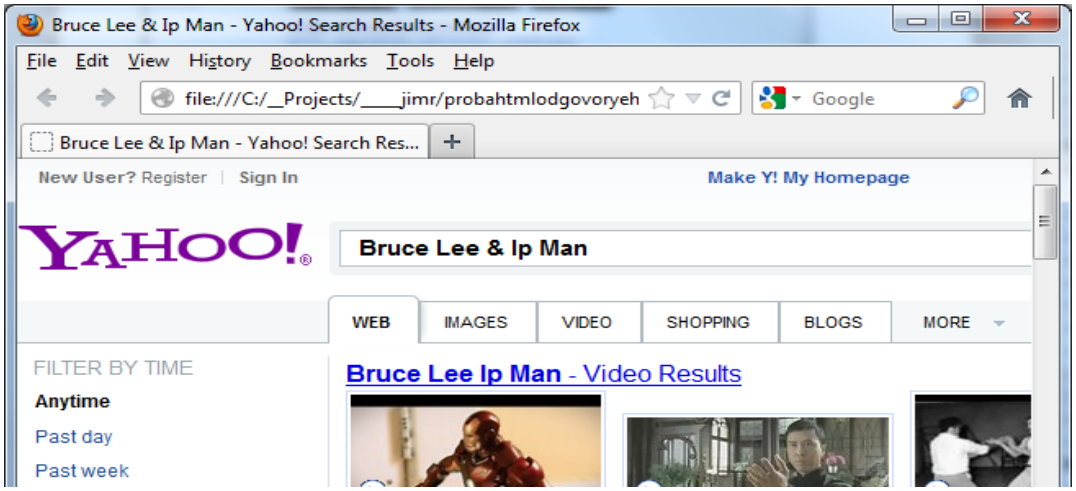
Slika 6.29. Slanje upita pretraživaču

Izlaz je kompletna html stranica odgovora:

```
<!doctype html>
<html lang="en-US">
<head>
  <meta http-equiv="content-type" content="text/html;
        charset=UTF-8">
...
```

```
...
</html>
```

Ako se ovaj odgovor snimi kao html datoteka, onda se pokretanjem ovog htmla dobija slika 6.30.



Slika 6.30. Klijentska strana za UDP komunikaciju

6.4.7.1. URL kodovanje

Da bi se postigla interoperabilnost između različitih platformi (Windows, Linux, Mac) potrebno je obezbediti isti pristup web-u gledano do na podskup ASCII karaktera. Ovaj podskup sadrži: velika slova: A-Z, mala slova: a-z, cifre: 0-9 i interpunkcijske znakove. Kodiranje se rešava tako da se svi karakteri koji nisu iz navedenog podskupa zamene sa kombinacijom procenta iza koga slede dve heksadecimalne znamenke.

Sledi program koji kodira upit prema prethodnom pravilu. Izlaz programa daje sledeće konverzije:

- razmak kodiran kao "+"
- "&" je kodiran kao "%26"
- ostali karakteri ostaju isti.

Za konverziju se korsiti `URLEncoder` klasa pomoću koje je moguće kodiranje stringa (URL-a) u različitim formatima. `URLDecoder` klasa obavlja suprotan proces. Navedene klase mogu se koristiti i za baratanje podacima HTML forme.

```
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;

public class FormatiranjeUpita{

    private String queryEngine;
    private StringBuilder query = new StringBuilder();
```

```
    public FormatiranjeUpita(String queryEngine) {
        this.queryEngine = queryEngine;
    }

    public String getEngine() {
        return this.queryEngine;
    }

    public void addQuery(String queryString,
                         String queryValue)
                                        throws Exception {
        query.append(
              queryString + "=" +
              URLEncoder.encode(queryValue, "UTF-8") + "&");
    }

    public String getQueryString() {
        return "?" + query.toString();
    }

    public static void main(String[] args)
                                         throws Exception{
        QueryStringFormatter formatter =
                new QueryStringFormatter(
                "http://www.google.com.au/search");
        formatter.addQuery("newwindow", "1");
        formatter.addQuery("q", "Nikola Tesla & Albert Einstein");
        System.out.println(formatter.getEngine()
                + formatter.getQueryString());
    }
}
```

Slika 6.31. Formatiranje upita za pretraživač

Izlaz:

http://www.google.com.au/search?newwindow=1&q=Nikola+Tesla+%26+Albert+Einstein&

Rezime poglavlja klijent-server komunikacija

U poglavlju klijent-server komunikacija obrađeno je pisanje klijent-server programa koji komuniciraju u računarskoj mreži korišćenjem TCP (*Transport Control Protocol*) i UDP (*User Datagram Protocol*) protokola.

Zadaci za proveru znanja iz poglavlja klijent-server komunikacija:

1. Napisati TCP komunikaciju klijenta i servera u kojoj klijent traži:
   - da mu server prosledi svoje vreme
   - da server zapamti poruku dobijenu od klijenta za drugog klijenta
   - da prosledi tekst poruke za klijenta (ako je bila neka poruka za klijenta).
   Program realizovati u GUI okruženju AWT, Swing, JavaFX.
2. Napisati porgram u kome klijent šalje fajl na server. Program realizovati u GUI okruženju AWT, Swing, JavaFX.
3. Napisati chat program u GUI okruženju AWT, Swing, JavaFX. Poruke klijenta vide svi priključeni klijenti.
4. Napisati JavaFX program za komunikaciju klijenta sa serverom tako da se pri komunikaciji nacrta redom :
   1. komunikacija - krug
   2. komunikacija - kao 1 ali i 4 manja kruga ortogonalno oko većeg kruga
   3. komunikacija - kao 2 ali i 4 manja kruga oko manjih krugova
   4. komunikacija - i tako redom
   Pomoć: razmislite o rekurzivnom rešenju.
5. Napisati program u GUI okruženju AWT, Swing, JavaFX, gde dva igrača (klijenta) igraju iks oks.

PRILOZI

Tabela A1.a. Događaji koje generišu AWT komponente

AWT komponenta	**Tipovi događaja koje komponenta može generisati**										
	action	adjustment	component	container	focus	item	Key	mouse	mouse motion	text	window
Button	X		X		X		X	X			
Canvas			X		X		X	X			
Checkbox			X		X	X	X	X			
CheckboxMenuItem						X					
Choice			X		X	X	X	X			
Component			X		X		X	X			
Container			X		X		X	X			
Dialog			X	X	X		X	X			X
Frame			X	X	X		X	X			X

Tabela A1.b. Događaji koje generišu AWT komponente

AWT komponenta	**Tipovi događaja koje komponenta može generisati**										
	action	adjustment	component	container	focus	item	Key	mouse	mouse motion	text	window
Label			X		X		X	X			
List	X		X		X	X	X	X			
MenuItem	X										
Panel			X	X	X		X	X			
Scrollbar		X	X		X		X	X			
ScrollPane			X	X	X		X	X			
TextArea			X		X		X	X		X	
TextComponent			X		X		X	X		X	
TextField	X		X		X		X	X		X	
Window			X	X	X		X	X			X

Tabela A2. Najčešće korišćeni događaji i odgovarajući osluškivači

Tip događaja	**Odgovarajući listener**
ActionEvent	ActionListener
AdjustmentEvent	AdjustmentListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener
KeyEvent	KeyListener
MouseEvent	MouseListener
WindowEvent	WindowListener
ItemEvent	ItemListener
TextEvent	TextListener

Tabela A3.a. Osluškivači, pripadni adapteri i metode

Interfejs	**Adapter Metodi**	
ActionListener	*nema*	actionPerformed
AdjustmentListener	*nema*	adjustmentValueChanged
ComponentListener	ComponentAdapter	componentHidden componentMoved componentResized componentShown
ContainerListener	ContainerAdapter	componentAdded componentRemoved
FocusListener	FocusAdapter	focusGained focusLost
ItemListener	*Nema*	itemStateChanged
KeyListener	KeyAdapter	keyPressed keyReleased keyTyped

Tabela A3.b. Osluškivači, pripadni adapteri i metode

MouseListener	MouseAdapter	mouseClicked mouseEntered mouseExited mousePressed mouseReleased
MouseMotionListener	MouseMotionAdapter	mouseDragged mouseMoved
MouseWheelListener	*nema*	mouseWheelMoved
TextListener	*nema*	textValueChanged
WindowListener	WindowAdapter	windowActivated windowClosed windowClosing windowDeactivated windowDeiconified windowIconified windowOpened

Tabela A4. Neke od najčešće korišćenih Swing GIU komponenti

Klasa	**Opis**
ButtonGroup	povezuje više radio dugmadi da rade zajedno; nije vidljiva komponenta
JButton	programsko dugme
JCheckBox	check box
JComboBox	combo box
JDialog	dijalog (prozor kome se ne može menjati veličina)
JFrame	prozor
JLabel	labela
JList	list box
JMenu	meni
JMenuBar	linija menija
JMenuItem	stavka menija
JOptionPane	prozor koji ispisuje kraću poruku (message box)
JPanel	komponenta koja je kontejner za druge komponente
JRadioButton	radio dugme
JTabbedPane	kartice (tabs); pojedine kartice se na ovu komponentu dodaju kao JPanel-i
JTextArea	višelinijsko polje za unos teksta (memo)
JTextField	jednolinijsko polje za unos teksta

LITERATURA

[1] Herbert Schildt : "*Java kompletan priručnik"*, prevod desetog izdanja, Mikroknjiga, Beograd, 2018.

[2] Dr. Edward Lavieri, Peter Verhas : "*Java 9"*, prevod prvog izdanja, Kompjuter biblioteka, Mikroknjiga, Beograd, 2018.

[3] Yakov Fain : "*Java 8 programiranje*", Kompjuter biblioteka, Beograd, 2015.

[4] Laslo Kraus : "*Rešeni zadaci iz programskog jezika JAVA JSE 8*", Akademska misao, Beograd, 2015.

[5] Laslo Kraus : "*Programski jezik Java sa rešenim zadacima JSE 8*", Akademska misao, Beograd, 2015.

[6] Bruce Eckel : "*Misliti na Javi"*, prevod 4. izdanja, Mikroknjiga, Beograd, 2007.

[7] Ivor Horton,"*Ivor Horton's Beginning Java™*", 2 JDK™ 5 Edition, Wiley Publishing, Inc, 2005.

[8] Branko Milosavljević, Vidaković M. : "*Java i Internet programiranje*", GInT, Novi Sad, 2002.

[9] Elliotte Rusty Harold : "*Java Network Programming, 3rd Edition*", O'Reilly Media, 2004.

[10] Brian Overland : "*Java in Plain English"*, MIS:Press, 1997.

[11] Perica Štrbac : "*Java i mrežno računarstvo*", Megatrend, Beograd, 2013.

[12] William A. Shay, "*Savremene komunikacione tehnologije i mreže*", Kompjuter biblioteka, Beograd, 2004.

[13] Matt Weisfeld : "*The Object-Oriented Thought Process*", Sams Publishing, 2000.

[14] Herbert Schildt : "*Java The Complete Reference, 8th Edition*", Oracle Press, 2011.

[15] Halabi, S., McPherson, D. : "*Internet Routing Architectures second edition*", Cisco press, 2002.

[16] Wetteroth, D. : "*OSI Reference Model for Telecommunications*", McGraw-Hill, New York, 2002.

[17] https://docs.oracle.com/javase/tutorial/ : "*The Java™ Tutorials*", Oracle, 2016.

[18] http://www.w3schools.com/, World Wide Web Consortium.

[19] https://docs.oracle.com/javafx/2/ , JavaFX, Oracle, 2018.

[20] Bigelow, J. S. : "*Računarske mreže*", CET, Beograd, 2004.

[21] Comer, E. D. : "*Povezivanje mreža – TCP/IP – Principi, protokoli i arhitekture*", prevod četvrtog izdanja, CET, Beograd, 2001.

[22] Habraken, J. : "*Osnove umrežavanja*", CET, Beograd, 2002.

[23] Joshua Bloch : "*Effective Java™, Second Edition*", Prentice Hall, 2008.

INDEKS POJMOVA

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z