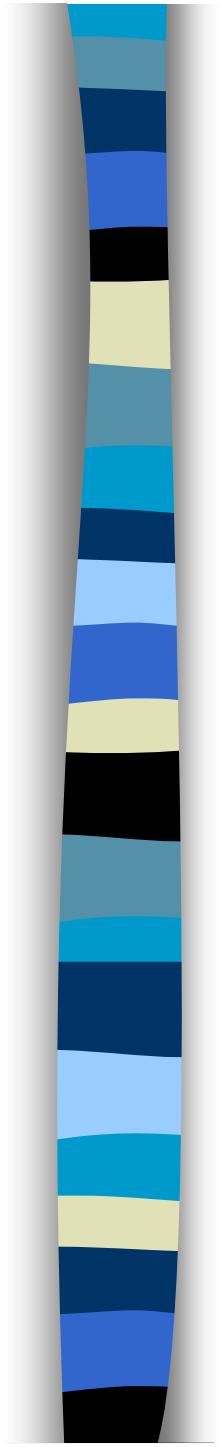


Razvoj vizuelnih aplikacija

Razvoj vizuelnih aplikacija

- Sve do sada razmatrane aplikacije su bile konzolne: korisnik je podatke unosio preko tastature, a poruke i rezultati izvršavanja su ispisivani na ekran.
- Međutim, važan aspekt pravljenja aplikacija predstavlja i kreiranje grafičkih korisničkih interfejsa, odnosno razvoj vizuelnih aplikacija.
- Java SE sadrži tri biblioteke za ovu namenu:
  - AWT (eng. Abstract Window Toolkit),
  - Swing i
  - JavaFX.

Razvoj vizuelnih aplikacija

- **AWT** je bila prva vizuelna biblioteka. Napisana je dosta brzo, s idejom da se iskoriste postojeće komponente u operativnim sistemima.
- Na primer, dugme u AWT biblioteci je samo omotač oko dugmeta koje već postoji u operativnom sistemu.
- Iako dosta brza, AWT biblioteka nije bila dovoljno dobra za razvoj kompleksnijih aplikacija, jer je sadržala samo komponente koje postoje na svim operativnim sistemima.
- Da bi se prevazišli nedostaci AWT-a, napisana je nova biblioteka, **Swing**, koja je objavljena u okviru J2SE 1.2.

Razvoj vizuelnih aplikacija

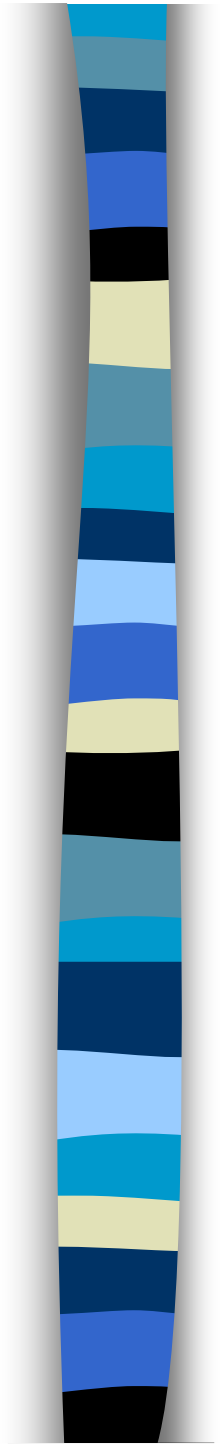
- **Swing** je u potpunosti napisan u Javi i sadrži veliki broj klasa i interfejsa za razvoj veoma kompleksnih vizuelnih aplikacija.
- U početku je biblioteka bila dosta spora, a Java aplikacije su vizuelno odudarale od tzv. **native aplikacija**.
- Međutim, performanse su vremenom popravljane, a uvedena je i mogućnost prilagođavanja izgleda konkretnom operativnom sistemu: Swing aplikacija bi, na primer, pod Windows-om izgledala kao standardna Windows, a pod Linux-om kao standardna Linux aplikacija.
- Swing je deo veće biblioteke pod nazivom Java Foundation Classes (JFC), koja uključuje još i originalni AWT i biblioteku Java 2D za rad sa grafikom u ravni.

Razvoj vizuelnih aplikacija

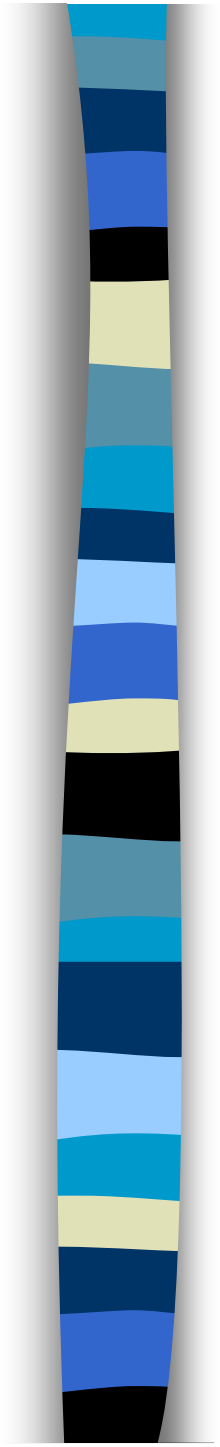
- 2008. godine Sun je objavio novi skript jezik pod nazivom **JavaFX**.
- Svrha novog jezika je bio razvoj tzv. ***bogatih internet aplikacija*** (eng. rich internet applications) vizuelnih aplikacija koje se izvršavaju u veb pretraživačima.
- JavaFX skript je prošao kroz nekoliko iteracija, da bi verzija 2.0 objavljena 2011. godine bila potpuno transformisana u Javu.
- Tj. ideja skript jezika je napuštena, a JavaFX postaje Java biblioteka koja nije deo Java SE, već se instalira dodatno. Ovo se ubrzo promenilo, i počev od Jave SE 7 update 6, JavaFX verzija 2.2 postaje integralni deo standardnog Java API.
- Naredna verzija JavaFX biblioteke nosi oznaku 8, koja se podudara sa aktuelnom verzijom Java SE.

Razvoj vizuelnih aplikacija

- Dugoročni plan je da JavaFX u potpunosti zameni Swing.
- JavaFX je modernija biblioteka, sa boljom podrškom za veb i mobilne platforme, i interno je bolje organizovana, te donekle lakša za upotrebu.
- Iako je Java već duže vreme jedna od najpopularnih programskih platformi, treba napomenuti da relativno mali broj programera zapravo razvija vizuelne aplikacije u Javi.
- Moć Jave je u serverskim aplikacijama, i Java EE tehnologijama, dok su za razvoj vizuelnih aplikacija popularniji drugi jezici i platforme (npr. HTML5 za veb).
- Međutim, poznavanje biblioteka Swing i JavaFX može pozitivno uticati na opšte obrazovanje programera, jer se koncepti na kojima se one zasnivaju koriste i u drugim vizuelnim bibliotekama (na primer: šablon MVC, programiranje vođeno događajima, itd.).



Biblioteka Swing



Prozori i apleti

- Klasa `JFrame`
- Klasa `JApplet`
- Pokretanje apleta
- Crtanje
- Uvod u interaktivne interfejse
- Raspoređivanje komponenti
- Model događaja
- Pregled *Swing* komponenti
- Animacije

Prozori i apleti

- Počev od verzije 1.2, Java poseduje biblioteku za kreiranje korisničkih interfejsa ***Swing.***
- Zbog kompatibilnosti sa starijim verzijama Jave, stari ***AWT*** paket je zadržan, a novi *Swing* je realizovan nasleđivanjem klasa iz starog paketa.
- Neke komponente starog *AWT* paketa se i dalje koriste direktno i zbog toga se gotovo uvek pri programiranju interfejsa uvoze oba paketa.
- **Apleti** su posebna vrsta Javinih programa koji se koriste kao delovi Internet prezentacija.
- Dok se prozori koriste za kreiranje grafičkih interfejsa koji podržavaju samostalne aplikacije, aplet se od običnog Java programa razlikuje po tome što se on ne može izvršavati samostalno, već mu je za to potreban neki drugi program, kao što je Internet brauzer.
- **Prozori** su u *Swing* biblioteci predstavljeni klasom **javax.swing.JFrame** dok se za kreiranje **apleta** koristi **javax.swing.Japplet.**

Klasa JFrame

```
java.lang.Object
      |
      +--java.awt.Component
           |
           +--java.awt.Container
                |
                +--java.awt.Window
                     |
                     +--java.awt.Frame
                          |
                          +--javax.swing.JFrame
```

- Prilikom programiranja grafičkog interfejsa, klasa **JFrame** se obično tretira samo kao **kontejner** na najvišem nivou hijerarhije, koji može da sadrži druge kontejnere ili specijalizovane komponente interfejsa kao što su **dugmad, labele, polja za upis teksta**, itd.
- Dodavanje komponenti koje će se pojaviti na površini prozora se obično vrši **u konstruktoru klase koja nasleđuje JFrame** pozivanjem metoda **add**, a prethodno je potrebno dobiti referencu na objekat sadržajnog okna prozora (engl.*content pane*)

Klasa `JFrame` - primer

Primer – kreiranje prozora

```
import javax.swing.*;
import java.awt.*;
public class MyFrame extends JFrame {
    public MyFrame() {
        getContentPane().add(new JLabel("Prozor!"));
    }
    public static void main(String[] args) {
        JFrame frame = new MyFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(200,150);
        frame.setVisible(true);
    }
}
```

Klasa JApplet

```
java.lang.Object
      |
      +--java.awt.Component
           |
           +--java.awt.Container
                |
                +--java.awt.Panel
                     |
                     +--java.applet.Applet
                          |
                          +--javax.swing.JApplet
```

- Slično kao u slučaju prozora, klasa **JApplet** obično služi samo kao kontejner za druge specijalizovane komponente interfejsa
- Prilikom kreiranja apleta najčešće je potrebno redefinisati jedan ili više metoda klase **JApplet:**
  - **`public void init()` -** Ovaj metod se poziva kada je potrebno inicijalizovati aplet (inicijalizacija se ne vrši u konstruktoru već u ovom metodu)
  - **`public void start()`** - Ovaj metod se poziva kada je potrebno početi sa izvršavanjem apleta. Pre nego što prvi put pozove ovaj metod, aplikacija će pozvati metod **`init`**.
  - **`public void stop()`** - Ovaj metod se koristi za zaustavljanje izvršavanja apleta.
  - **`public void destroy()`** - Ovim metodom se aplet uništava od strane aplikacije u kojoj se aplet izvršava.

Klasa `JApplet` - primer

Primer – kreiranje apleta

```
import javax.swing.*;
import java.awt.*;

public class MyApplet extends JApplet {
  public void init() {
    getContentPane().add(new JLabel("Aplet!"));
  }
}
```

- Dodavanje komponenti se obavlja u redefinisanom metodu **init**, pozivanjem metoda **add**, ali je prethodno potrebno dobiti referencu na objekat sadržajnog okna
- Apleti ne moraju da imaju implementiran metod **main**, već se kreiranje odgovarajuće intsance apleta i njegovo prikazivanje na ekranu obično vrši od strane sistema

Pokretanje apleta

1. Nakon prevođenja fajla u kojem se nalazi tekst apleta, napravljeni aplet možemo ugraditi u HTML

Primer - HTML stranica sa ugrađenim apletom `MyApplet`

```
<HTML>
  <HEAD>
    <TITLE> Nasa web stranica </TITLE>
  </HEAD>
  <BODY>
    Ovde je rezultat naseg programa:
    <APPLET CODE="MyApplet.class" WIDTH=150 HEIGHT=25>
    </APPLET>
  </BODY>
</HTML>
```

2. Pokretanje apleta se može ostvariti i korišćenjem Java alata *appletviewer*

- Pošto *appletviewer* traži **<applet>** tagove da bi pokrenuo aplete **<applet>** tag se vrlo često ubacuje na početak izvornog fajla (na primer **MyApplet.java)** pod komentarom:

```
\\<applet code="MyApplet.class" width=200 height=100></applet>
```

- Na taj način se može pozvati ***appletviewer MyApplet.java***, a da se izbegne kreiranje HTML fajla

Pokretanje sa komandne linije

- Pri testiranju apleta često brže i lakše pokrenuti rezultujuću aplet-aplikaciju sa komandne linije nego pokretanjem brauzera ili *appletviewer*-a.
- Da bi se kreirao aplet koji je moguće pokrenuti sa komandne linije, apletu je potrebno dodati main metod koji pravi instancu apleta unutar prozora.
- U telu metoda main aplet se mora eksplicitno inicijalizovati i pokrenuti pošto u ovom slučaju nema brauzera, koji pored metoda init i start, takođe poziva metode stop i destroy.
- Pa ipak, u većini situacija ovakva realizacija je prihvatljiva.
- Ako ipak nedostatak poziva ovih metoda predstavlja problem, i ti pozivi se mogu uvrstiti u program.

Pokretanje sa komandne linije

- Pri testiranju apleta često brže i lakše pokrenuti rezultujuću aplet-aplikaciju sa komandne linije nego pokretanjem brauzera ili *appletviewer*-a

Primer - instanca apleta unutar prozora

```
// <applet code="MyApplet.class" width=100 height=50></applet>
import javax.swing.*;
import java.awt.*;

public class MyApplet extends JApplet {
  public void init() {
    getContentPane().add(new JLabel("Aplet!"));
  }

  public static void main(String[] args) {
    JFrame frame = new JFrame("MyApplet");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JApplet applet = new MyApplet();
    applet.init();
    applet.start();

    frame.getContentPane().add(applet);
    frame.setSize(100,50);
    frame.setVisible(true);
  }
}
```

Crtanje (1/2)

- Svaka komponenta je odgovorna za svoje iscrtavanje na ekranu.
- Standardne komponente, se samo dodaju na površinu apleta ili prozora, a njihovo iscrtavanje se događa automatski.
- Iscrtavanje nikada ne vrši direktno na površini apleta ili neke druge komponente najvišeg nivoa *Swing* hijerarhije - obično se definiše zasebna komponenta koja služi kao površina za crtanje, najčešće podklasa klase **javax.swing.Jpanel.**
- Svaka komponenta u *Swing* bilioteci svoj sadržaj iscrtava u metodu **paintComponent(Graphics g).**
- Da bi se kreirala proizvoljno iscrtana površina, potrebno je naslediti klasu **JPanel** i redefinisati njen metod **paintComponent.**
- Pravo pozivanja metoda **paintComponent** je rezervisano za sistem, osvežavanje prikaza komponente pozivom metoda **repaint().**
- Metod **paintComponent** ima jedan formalni parametar tipa **java.awt.Graphics** koji predstavlja grafički kontekst pomoću kojeg se vrši iscrtavanje proizvoljnih geometrijskih oblika, teksta ili slika.

Crtanje (2/2)

- Referenca na grafički objekat se po potrebi može dobiti i izvan ovog metoda pozivom metoda komponente **getGraphics()**
- Površina komponente je pravougaona matrica tačaka (piksela), a pozicija svakog piksela je određena koordinatama – koordinatni početak postavljen je u gornjem levom uglu komponente
- Klasa **Graphics** poseduje veliki broj metoda za crtanje različitih geometrijskih oblika koji se iscrtavaju u tekućoj boji grafičkog objekta (inicijalno crna), ali se ova boja može promeniti pozivom metoda grafičkog objekta **setColor**
- Neki od tipičnih metoda za crtanje su:
  - `drawString(String str, int x, int y)`
  - `drawLine(int x1, int y1, int x2, int y2)`
  - `drawRect(int x, int y, int width, int height)`
  - `fillRect(int x, int y, int width, int height)`
  - `drawOval(int x, int y, int width, int height)`
  - `fillOval(int x, int y, int width, int height)`

Uvod u interaktivne interfejse

- Za razliku od proceduralnih programa u kojima se naredbe izvršavaju sekvencijalno, programi koji opisuju interakciju sa korisnikom pomoću grafičkih interfejsa su asinhroni.
- Program mora biti spreman da odgovori na razne vrste događaja koji se mogu desiti u bilo koje vreme i to redosledom koji program ne može da kontroliše. Osnovne tipove događaja čine oni koji su generisani mišem ili tastaturom.
- Programiranje vođeno događajima (engl. ***event-driven programming***) - sistemski dodeljena nit se izvršava u petlji čekajući na akcije korisnika.
- Programiranje interaktivnih prozorskih aplikacija se svodi na raspoređivanje komponenti u okviru prozora, a zatim i na definisanje metoda koji bi služili za obradu događaja koje te komponente mogu da proizvedu.

Uvod u interaktivne interfejse - primer (1/2)

Primer - aplet koji interaktivno menja sadržaj labele

```
// <applet code="Dugme.class" width=300 height=75></applet>
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class Dugme extends JApplet {
  private JButton
    d1 = new JButton("Dugme 1"),
    d2 = new JButton("Dugme 2");
  private JLabel txt = new JLabel("Pocetak");
```

Uvod u interaktivne interfejse - primer (2/2)

```
private ActionListener dl = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      String name = ((JButton)e.getSource()).getText();
      txt.setText(name);
    }
  };
  public void init() {
    d1.addActionListener(dl);
    d2.addActionListener(dl);
    Container cp = getContentPane();
    cp.setLayout(new FlowLayout());
    cp.add(d1);
    cp.add(d2);
    cp.add(txt);
  }
  public static void main(String[] args) {
    JFrame frame = new JFrame("Dugme");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JApplet applet = new Dugme();
    applet.init();
    applet.start();
    frame.getContentPane().add(applet);
    frame.setSize(300,75);
    frame.setVisible(true);
  }
}
```

Raspoređivanje komponenti

- Raspoređivači (engl. ***layout managers***) su klase koje specificiraju kako bi komponente trebalo da budu raspoređene.
- Raspoređivači se takođe prilagođavaju dimenzijama apleta, odnosno prozora, tako da se veličina i raspored komponenti menjaju sa promenom veličine prozora.
- Svaki objekat klase **Container** sadrži metod **setLayout** koji omogućava postavljanje proizvoljnog raspoređivača njegovih komponenti.
- Pozivom metoda **getContentPane()** za objekte klasa **JApplet**, **JFrame**, **JWindow** i **JDialog** dobija se referenca na "sadržajno okno" (engl. *content pane*), koji može da sadrži i prikazuje različite komponente indirektno sadržane na površini objekata ovih klasa.
- Objekti drugih klasa, poput klase **JPanel** koja je takođe podklasa klase **Container**, mogu da sadrže i prikazuju komponente direktno bez posredstva sadržajnog okna.
- Pre nego što se neki prozor prikaže na ekranu, mora mu se dodeliti inicijalna veličina ili se mora pozvati njegov metod **pack().**

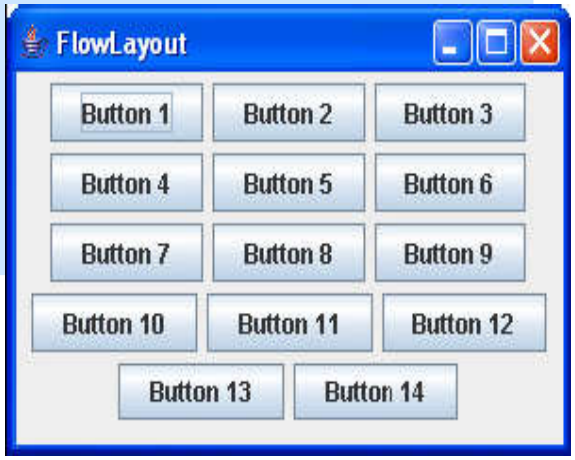
FlowLayout

- Inicijalni raspoređivač klase **JPanel**, raspoređuje komponente na način na koji se prelama tekst.
- Komponente seređaju sa leva na desno redom kojim se dodaju na provršinu kontejnera, sve dok njihova ukupna širina ne prelazi širinu kontejnera.
- Ako se koristi konstruktor raspoređivača bez argumenata, redovi će biti centrirani, a horizontalno i vertikalno rastojanje između komponenti će biti po 5 piksela.
- Možemo koristiti i konstruktor **FlowLayout(int align, int hgap, int vgap)**, gde se za vrednosti poravnanja mogu koristiti neke od vrednosti **FlowLayout.LEFT, FlowLayout.RIGHT** ili **FlowLayout.CENTER**, a rastojanje se izražava u pikselima.
- Za postavljanje komponenti u kontejner koristimo verziju metoda **add** koji prima samo jedan argument – komponentu koja se postavlja.

FlowLayout - primer

```
// <applet code="FlowLayout1.class" width=300 height=250></applet>
import javax.swing.*;
import java.awt.*;

public class FlowLayout1 extends JApplet {
  public void init() {
    Container cp = getContentPane();
    cp.setLayout(new FlowLayout());
    for(int i = 1; i < 15; i++)
      cp.add(new JButton("Button " + i));
  }
  public static void main(String[] args) {
    JFrame frame = new JFrame("FlowLayout1");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JApplet applet = new FlowLayout1();
    applet.init();
    applet.start();
    frame.getContentPane().add(applet);
    frame.setSize(300,250);
    frame.setVisible(true);
  }
}
```



Podešavanje veličine dugmeta

```
JButton b = new JButton("Button " + i);
b.setPreferredSize(new Dimension(88,26));
cp.add(b);
```

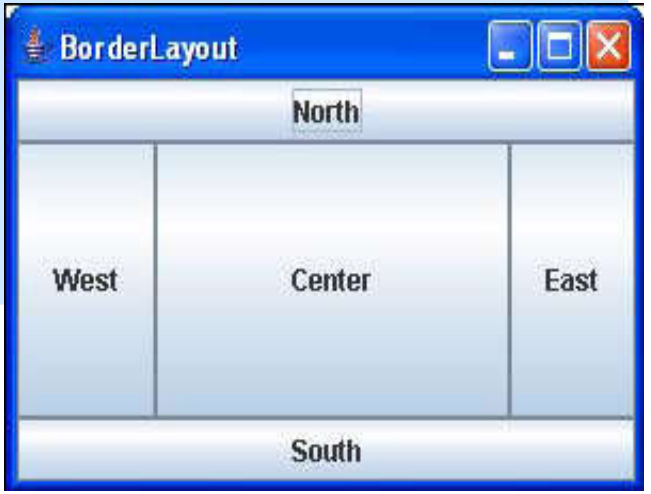
BorderLayout

- Inicijalni raspoređivač sadržajnog okna apleta i prozora
- Bez ikakvih dodatnih podešavanja, komponenta koja se dodaje na površinu apleta ili prozora biće centralno postavljena i razvučena (korišćenjem metoda **add** sa jednim argumentom)
- Korišćenjem metoda **add** koji prima dva argumenta(komponenta koja se postavlja, područje na kojem komponenta treba da bude smeštena), dobijamo nešto širi koncept raspoređivanja komponenti, po kojem je površina kontejnera podeljena na 4 ivična područja i jedan centralni deo
- Konstante za pozicioniranje: **BorderLayout.NORTH, BorderLayout.SOUTH, BorderLayout.EAST, BorderLayout.WEST, BorderLayout.CENTER**
- Konstruktorom **BorderLayout(int hgap, int vgap)** određuje se veličina praznog prostora između komponenti

BorderLayout - primer

```
// <applet code="BorderLayout1.class" width=300 height=250></applet>
import javax.swing.*;
import java.awt.*;

public class BorderLayout1 extends JApplet {
  public void init() {
    Container cp = getContentPane();
    cp.add(new JButton("North"), BorderLayout.NORTH);
    cp.add(new JButton("South"), BorderLayout.SOUTH);
    cp.add(new JButton("East"), BorderLayout.EAST);
    cp.add(new JButton("West"), BorderLayout.WEST);
    cp.add(new JButton("Center"), BorderLayout.CENTER);
  }
  public static void main(String[] args) {
    JFrame frame = new JFrame("BorderLayout1");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JApplet applet = new BorderLayout1();
    applet.init();
    applet.start();
    frame.getContentPane().add(applet);
    frame.setSize(300,250);
    frame.setVisible(true);
  }
}
```



GridLayout

- Formira tabelu međusobno jednakih komponenti sa odgovarajućim brojem vrsta i kolona.
- Koristi se verzija metoda **add** sa jednim argumentom.
- Komponente će u tabeli biti raspoređene redom kojim se postavljaju na površinu kontejnera tako što se tabela popunjava po vrstama, od gornjih ka donjim, pri čemu se svaka vrsta popunjava sa leva na desno.
- Broj vrsta i kolona se zadaje konstruktorom **GridLayout(int rows, int cols)**i u tom slučaju komponente se maksimalno zbijaju.
- Za određivanje horizontalnog i vertikalnog rastojanja između komponenti koristi se konstruktor **GridLayout(int rows, int cols, int hgap, int vgap).**

GridLayout - primer

```
// <applet code="GridLayout1.class" width=300 height=250></applet>
import javax.swing.*;
import java.awt.*;

public class GridLayout1 extends JApplet {
  public void init() {
    Container cp = getContentPane();
    cp.setLayout(new GridLayout(5,3));
    for(int i = 1; i < 15; i++)
      cp.add(new JButton("Button " + i));
  }
  public static void main(String[] args) {
    JFrame frame = new JFrame("GridLayout1");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JApplet applet = new GridLayout1();
    applet.init();
    applet.start();
    frame.getContentPane().add(applet);
    frame.setSize(300,250);
    frame.setVisible(true);
  }
}
```

GridBagLayout

- Sličan raspoređivaču tipa **GridLayout** po tome što od postavljenih elementata formira tabelu.
- Omogućava izvođenje složenijih zahvata, pošto redovi ne moraju biti iste visine, kolone ne moraju biti iste širine i komponente mogu da se prostiru preko više vrsta i/ili kolona.
- Posebna klasa **GridBagConstraints** omogućava zadavanje pozicija komponenti, broj vrsta i/ili kolona preko kojih se prostiru, kao i mnoge druge osobenosti postavljenih komponenti.
- Omogućava odličnu kontrolu u odlučivanju kako će delovi prozora biti raspoređeni i kako se reogranizuju kada se veličina prozora promeni.
- Najkomplikovaniji rapoređivač - prevashodno namenjen za automatsko generisanje koda od strane vizuelnih alata.

Apsolutno pozicioniranje

- Ukoliko je raspoređivač kontejnera postavljen na **null**, možemo dodeljivati apsolutne veličine i pozicije komponentama.
- Raspoređivač kontejnera se uklanja pozivom **setLayout(null)** nakon čega je odgovornost za raspoređivanje komponenti u kontejneru prepuštena programeru.
- Za svaku komponentu postavljenu na površinu takvog kontejnera, može se pozvati metod **setBounds(int x, int y, int width, int height)** koji postavlja gornji levi ugao komponente na poziciju **(x,y)** i njenu širinu i visinu na zadate vrednosti.
- Metod **setBounds** bi trebalo pozivati samo za komponente koje pripadaju kontejneru bez raspoređivača.

BoxLayout i klasa Box

- Omogućava postavljanje komponenti horizontalno u jednoj vrsti ili vertikalno u jednoj koloni.
- Uvek forsira minimalnu veličinu komponenti (za razliku od **GridLayout**-a), tako da se veličine postavljenih komponenti mogu razlikovati (kao **FlowLayout**).
- Obično se koristi preko instanci klase **javax.swing.Box** - dva statička metoda ove klase **Box.createHorizontalBox()** i **Box.createVerticalBox()** umesto konstruktora služe za kreiranje odgovarajućih instanci ove klase.
- Metodom **Box.createHorizontalStrut(int width)** se formira horizontalni razmak širine fiksnih ***width*** piksela. Analogno se koristi i **Box.createVerticalStrut(int height)** .
- Metodom **Box.createHorizontalGlue()** se formira razmak koji komponente udaljava na maksimalno moguće horizontalno rastojanje,a ukoliko postoji više ovakvih razmaka ukupan raspoloživi prostor se deli podjednako na sve prisutne razmake tog tipa. Isto važi i za metod **Box.createVerticalGlue().**
- Postavljanje komponenti se vrši pomoću metoda **add** sa jednim argumentom.

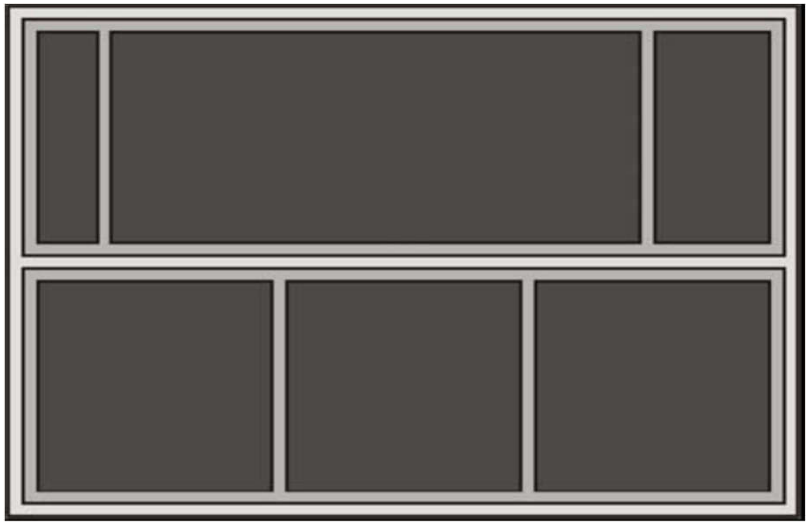
BoxLayout i klasa Box - primer

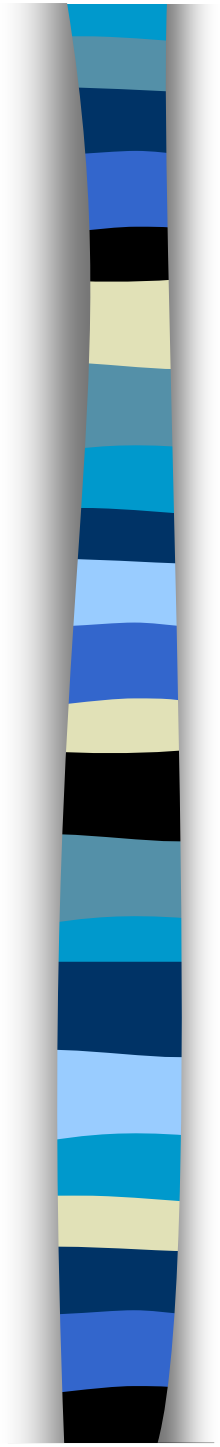
```
// <applet code="Box1.class" width=300 height=250></applet>
import javax.swing.*;
import java.awt.*;

public class Box1 extends JApplet {
  public void init() {
    Box hbox = Box.createHorizontalBox();
    hbox.add(new JButton("Prvi"));
    hbox.add(new JButton("Drugi"));
    hbox.add(Box.createHorizontalStrut(10));
    hbox.add(new JButton("Treci"));
    hbox.add(Box.createHorizontalGlue());
    hbox.add(new JButton("Cetvrti"));
    getContentPane().add(hbox, BorderLayout.NORTH);
  }
  public static void main(String[] args) {
    JFrame frame = new JFrame("Box1");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JApplet applet = new Box1();
    applet.init();
    applet.start();
    frame.getContentPane().add(applet);
    frame.setSize(300,250);
    frame.setVisible(true);
  }
}
```

Kombinovanje raspoređivača grupisanjem elemenata

- Standardna klasa **JPanel** – kontejner (može da sadržava druge komponente) i komponenta (može se postaviti na površinu apleta ili drugog panela)
- Time je omogućeno proizvoljno složeno ugneždavanje komponenti, pri čemu svaki panel može posedovati proizvoljnog raspoređivača svojih komponenti *(na slici – tri panela sa šest komponenti)*





Model događaja

Model događaja (1/2)

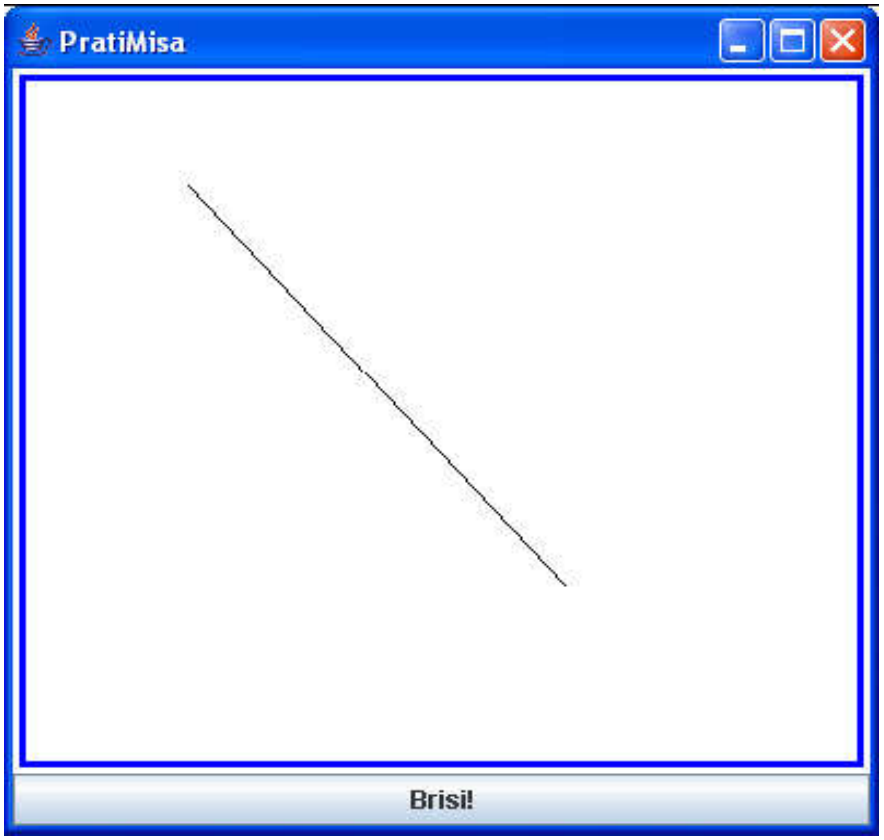
- Po modelu događaja koji je implementiran u Javi, komponente imaju sposobnost da opažaju događaje i emituju signale o njihovom pojavljivanju.
- **Događaji** su predstavljeni objektima - različiti tipovi događaja su predstavljeni objektima različitih klasa (npr. **MouseEvent** - kada korisnik klikne mišem).
- Kada je objekat događaja kreiran, on se prosleđuje kao parametar odgovarajućem metodu prijavljenog osluškivača koji služi za obradu datog događaja.
- Detekcija događaja se postiže osluškivanjem od strane objekata koji se nazivaju osluškivačima.
- **Osluškivači** zato moraju imati implementirane metode koji određuju reakciju objekta na obaveštenje o nastalom događaju.
- Svi ti metodi kao svoj argument primaju objekat istog osnovnog tipa (objekat događaja – npr. MouseEvent).

Model događaja (2/2)

- Na primer, **MouseListener** ima zasebno definisane metode koji se pozivaju kada korisnik pritisne taster miša, kada ga otpusti, kada pokazivač miša dođe na površinu komponente, kada pokazivač miša ode sa površine komponente, itd.
- Svaki osluškivač je objekat neke klase koja mora da implementira odgovarajući interfejs.
- Ime interfejsa je oblika **XxxListener** (npr. **MouseListener**, **KeyListener**)
- "Xxx" predstavlja tip događaja koji se osluškuje
- Sve što treba da uradimo jeste da kreiramo objekat osluškivača određenog tipa i prijavimo ga komponenti koja ima sposobnost da opaža događaje tog tipa.
- Svaka komponenta *Swing* biblioteke, u zavisnosti od toga koje tipove događaja podržava, poseduje odgovarajuće metode oblika **addXxxListener** i **removeXxxListener** koji služe za prijavljivanje i odjavljivanje osluškivača.
- Sve klase i interfejsi koji služe za rad sa događajima se nalaze u paketu **java.awt.event.**

Primer - interaktivno crtanje duži (1/6)

- Klikom miša se određuje početna tačka duži, pomeranjem miša se duž iscrtava, a novim klikom se odredjuje krajnja tačka, nakon čega nacrtana duž ostaje fiksirana. Aplet ilustruje upotrebu čak šest tipova osluškivača.



Aplet poseduje dve komponente:

- površinu za crtanje duži

- dugme koje služi za brisanje eventualno nacrtane duži.

Primer - interaktivno crtanje duži (2/6)

```
// <applet code="PratiMisa.class" width=400 height=400></applet>
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PratiMisa extends JApplet {

  Display display;
  JButton dugme;

  public void init() {
    display = new Display();
    dugme = new JButton("Brisi!");
    // osluškivač koji reaguje na pritisak dugmeta
    ActionListener al = new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        display.resetujTacke();// postavlja koordinate na početnu vred.
        display.repaint();// osvežava sadržaj na ekranu
      }
    };
    dugme.addActionListener(al);
    Container cp = getContentPane();
    cp.add(display, BorderLayout.CENTER);
    cp.add(dugme, BorderLayout.SOUTH);
  }
```

Primer - interaktivno crtanje duži (3/6)

```
class Display extends JPanel {
  int
    startx = -1, // koordinate pocetne tacke duzi
    starty = -1,
    mousex = -1, // koordinate krajnje tacke duzi
    mousey = -1;
  boolean focussed = false; //da li komponenta ima fokus da bi mogla da
                            // opaža događaje sa tastature
  Color lineColor = Color.black; // boja kojom se duž iscrtava

  void resetujTacke() {

    startx = starty = mousex = mousey = -1;
  }

  public void paintComponent(Graphics g) {
    // bojenje pozadine sa okvirom, beli okvir, plavi okvir, bela
    // unutrašnjost
    g.setColor(Color.white);
    g.fillRect(0,0,getSize().width,getSize().height);
    g.setColor(focussed ? Color.blue : Color.gray);
    g.fillRect(3,3,getSize().width-6,getSize().height-6);
    g.setColor(Color.white);
    g.fillRect(6,6,getSize().width-12,getSize().height-12);
    // crtanje duzi
    g.setColor(lineColor);
    g.drawLine(startx, starty, mousex, mousey);
  }
```

Primer - interaktivno crtanje duži (4/6)

```
public Display() {

  // osluškivač koji vodi računa o pritisku na taster miša
  MouseListener ml = new MouseListener() {
    public void mousePressed(MouseEvent evt) {
      requestFocus();
      startx = mousex = evt.getX();
      starty = mousey = evt.getY();
    }
    public void mouseReleased(MouseEvent evt) {}
    public void mouseEntered(MouseEvent evt) {}
    public void mouseExited(MouseEvent evt) {}
    public void mouseClicked(MouseEvent evt) {}
  };

  // osluškivač koji vodi računa o pokretima miša
  MouseMotionListener mml = new MouseMotionListener() {
    public void mouseDragged(MouseEvent evt) {
      mousex = evt.getX();
      mousey = evt.getY();
      repaint();
    }
    public void mouseMoved(MouseEvent evt) {}
  };
```

Primer - interaktivno crtanje duži (4/6)

```
// Osluškivač koji vodi računa da li komponenta ima fokus

FocusListener fl = new FocusListener() {
  public void focusGained(FocusEvent evt) {
    focussed = true;
    repaint();
  }
  public void focusLost(FocusEvent evt) {
    focussed = false;
    repaint();
  }
};

// Nakon korigovanja vrednosti forsira se osvežavanje
// sadržaja na ekranu da bi se nastala situacija odrazila
// u boji okvira
```

Primer - interaktivno crtanje duži (5/6)

```
      // Osluškivač koji vodi računa o tipkama tastature
      KeyListener kl = new KeyListener() {
        public void keyTyped(KeyEvent evt) {
          char ch = evt.getKeyChar();
          if (ch == 'B' || ch == 'b') {
            lineColor = Color.blue;
            repaint();
          } else if (ch == 'G' || ch == 'g') {
            lineColor = Color.green;
            repaint();
          } else if (ch == 'R' || ch == 'r') {
            lineColor = Color.red;
            repaint();
          } else if (ch == 'K' || ch == 'k') {
            lineColor = Color.black;
            repaint();
          }
        }
        public void keyPressed(KeyEvent evt) {}
        public void keyReleased(KeyEvent evt) {}
      };
      addMouseListener(ml);
      addMouseMotionListener(mml);
      addFocusListener(fl);
      addKeyListener(kl);
    }
  }
```

Primer - interaktivno crtanje duži (6/6)

```
private static JApplet applet;
public static void main(String[] args) {
    JFrame frame = new JFrame("PratiMisa");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    applet = new PratiMisa();
    applet.init();
    applet.start();
    // implementacija mehanizma za pokretanje apleta sa komandne linije
    WindowListener wl = new WindowListener() {
      // čeka na zatvaranje prozora da bi zaustavio applet
      public void windowClosing(WindowEvent e) {
        applet.stop();
        applet.destroy();
      }
      public void windowActivated(WindowEvent e) {}
      public void windowClosed(WindowEvent e) {}
      public void windowDeactivated(WindowEvent e) {}
      public void windowDeiconified(WindowEvent e) {}
      public void windowIconified(WindowEvent e) {}
      public void windowOpened(WindowEvent e) {}
    };
    frame.addWindowListener(wl);
    frame.getContentPane().add(applet);
    frame.setSize(400,400);
    frame.setVisible(true);
  }
}
```

Opšti postupak

- Postupak rada sa događajima i osluškivačima može biti sumiran na sledeći način:
  - Ubaciti uvoznu specifikaciju **import java.awt.event.*** na početak programa.
  - Obezbediti klasu koja će implementirati odgovarajući interfejs osluškivača, kao što je, na primer, **MouseListener**. Najelegantnija je primena anonimnih unutrašnjih klasa.
  - Obezbediti definicije metoda koji su deklarisani u interfejsu.
  - Kreirati objekat date klase i prijaviti ga komponenti koja će emitovati događaje pozivom odgovarajućeg metoda komponente, kao što je, na primer, **addMouseListener.**

Adapteri

- Prilikom implementacije osluškivača često nam je potrebno da definišemo samo mali broj od ukupnog broja metoda deklarisanih u okviru interfejsa.
- Pravila jezika nalažu da u slučaju kada klasa implementira neki interfejs, ona mora obezbediti definicije svakog od deklarisanih metoda. Čak i ako nisu potrebne sve metode interfejsa, moramo obezbediti njihove definicije makar sa praznim telima metoda.
- Zato su, u slučaju interfejsa koji deklarišu više od jednog metoda, dodatno obezbeđene odgovarajuće apstraktne klase – **adapteri**.
- Ove klase implementiraju odgovarajuće interfejse osluškivača, definišu sve metode tražene interfejsom, a ti metodi imaju prazna tela i nisu deklarisani kao apstraktni.
- To nam omogućava da kreiramo osluškivače kao podklase adaptera, i da na taj način obezbedimo samo onoliko definicija metoda koliko želimo, umesto da implementiramo sve metode interfejsa.

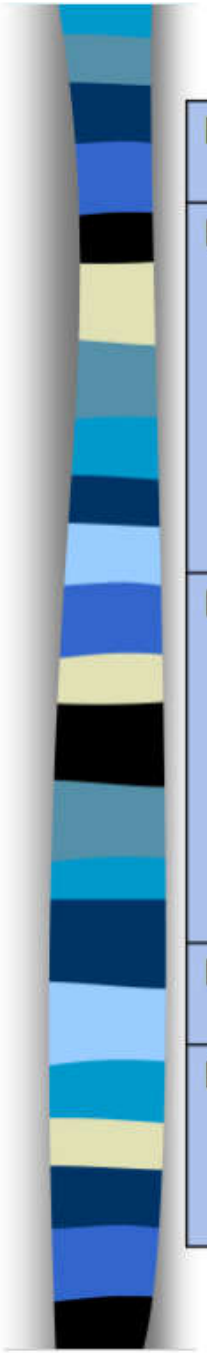
Adapteri

Primer – kreiranje osluškivača tipa `WindowListener`

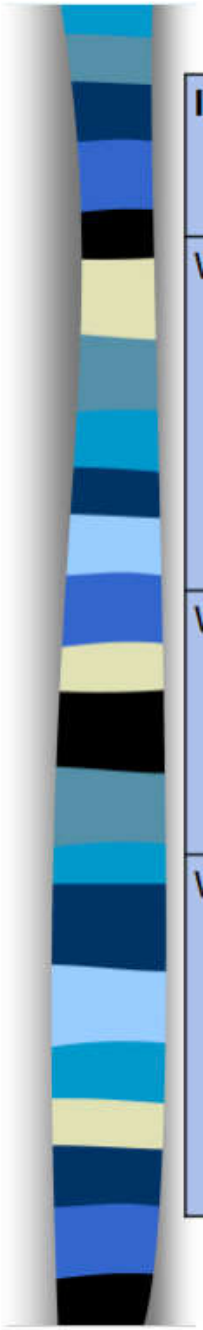
```
WindowListener wl = new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        applet.stop();
        applet.destroy();
    }
};
```

- Nedostatak korišćenja adaptera
- S obzirom na to da metodi nisu deklarisani kao apstraktni, upravo zato da ne bismo imali obavezu da ih implementiramo, kompajler neće vršiti proveru da li je metod implementiran ili ne, pa je mogućnost greške veća.
- Npr. ako umesto windowClosing napišemo WindowClosing, taj metod će se smatrati potpuno novim, a njegov kod neće služiti ničemu – neće se pozivati u sklopu mehanizma reagovanja na događaje kao što bismo očekivali.

Interfejs	Metode interfejsa	Događaj i njegove metode	Komponenta koja generiše događaj
ActionListener	actionPerformed	**ActionEvent** getActionCommand getModifiers	AbstractButton JComboBox JTextField Timer
AdjustmentListener	adjustmentValueChanged	**AdjustmentEvent** getAdjustable getAdjustmentType getValue	JScrollbar
ItemListener	itemStateChanged	**ItemEvent** getItem getItemSelectable getStateChange	AbstractButton JComboBox
FocusListener	focusGained focusLost	**FocusEvent** isTemporary	Component



Interfejs	Metode interfejsa	Događaj i njegove metode	Komponenta koja generiše događaj
KeyListener	keyPressed keyReleased keyTyped	**KeyEvent** getKeyChar getKeyCode getKeyModifiersText getKeyText isActionKey	Component
MouseListener	mousePressed mouseReleased mouseEntered mouseExited mouseClicked	**MouseEvent** getClickCount getX getY getPoint TranslatePoint	Component
MouseMotionListener	mouseDragged mouseMoved	**MouseEvent**	Component
MouseWheelListener	mouseWheelMoved	**MouseWheelEvent** getWheelRotation getScrollAmount	Component



Interfejs	**Metode interfejsa**	**Događaj i njegove metode**	**Komponenta koja generiše događaj**
WindowListener	windowClosing windowOpened windowIconified windowDeiconified windowClosed windowActivated windowDeactivated	**WindowEvent** getWindow	Window
WindowFocusListener	windowGainedFocus windowLostFocus	**WindowEvent** getOppositeWindow	Window
WindowStateListener	WindowStateChanged	**WindowEvent** getOldState getNewState	Window

Teorijske vežbe

Objektno-orjentisano programiranje

Izuzetak

- **Exception (exceptional event)**
  - Događaj koji narušava normalan tok izvršavanja programa i signalizira da se desila neka greška
- Dva tipa izuzetaka u Javi
  - **checked – izuzeci koji se moraju obavezno obraditi ili proslediti**
    - Program ne prolazi proces kompajliranja ako checked izuzetak nije obrađen/prosleđen
    - IOException, FileNotFoundException
  - **unchecked – izuzeci koji se ne moraju odbraditi**
    - Program prolazi proces kompajliranja ako unchecked izuzetak nije obrađen
    - ArrayIndexOutOfBoundsException, NegativeArraySizeException, NullPointerException, IllegalArgumentException, ArithmeticException, ClassCastException, NumberFormatException
- Svi izuzeci su klase koje nasleđuju (direktno ili indirektno) klasu Exception
  - Unchecked izuzeci su klase koje nasleđuju klasu RunTimeException koja nasleđuje klasu Exception

Obrada izuzetaka

- try – catch – finally blok

```
try {
        // neki kod koji može da uzrokuje pojavu izuzetaka
         // ExceptionClass1, ExceptionClass2, … ExceptionClassN
} catch (ExceptionClass1 name1) {
         // obrada izuzetka ExceptionClass1
} catch (ExceptionClass2 name2) {
        // obrada izuzetka ExceptionClass2
}
…
} catch (EceptionClassN nameN) {
        // obrada izuzetka ExceptionClassN
} finally {
        // kod koji se izvršava na kraju desio se neki izuzetak ili ne
}
```

- Mora postojati bar jedan catch blok ili tačno jedan finally block. Catch blokova može biti i više
  - Redosled je bitan ako postoji hijerarhija nasleđivanja među izuzecima
  - Od specifičnijih ka apstraktnijim izuzecima
- Finally blok je opcion (ne mora postojati) ukoliko postoji bar jedan catch blok.

Unchecked izuzeci

- Unchecked izuzeci su uglavnom posledica semantičkih grešaka (bugova) u programu
- **Popraviti bug ako postoji, a ne obrađivati unchecked izuzetak**

Exception1.java

```
import java.util.Scanner;

public class Exception1 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int a = s.nextInt();
        int b = s.nextInt();
        int c = a / b;
        System.out.println(c);
        s.close();
    }
}
```

Problems @ Javadoc Declaration Console Call Hierarchy Synchrc

```
<terminated> Exception1 [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (Dec 5,
10
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Exception1.main(Exception1.java:9)
```

Loše rešenje

Exception2.java

```
import java.util.Scanner;

public class Exception2 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);

        try {
            int a = s.nextInt();
            int b = s.nextInt();
            int c = a / b;
            System.out.println(c);
        } catch (ArithmeticException ae) {
            System.out.println("Deljenje nulom");
        }

        s.close();
    }
}
```

Problems @ Javadoc Declaration Console Call H

```
<terminated> Exception2 [Java Application] C:\Program Files\Java\jre1.8.0_65
10
0
Deljenje nulom
```

Dobro rešenje

```
import java.util.Scanner;

public class DobroResenje {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int a = s.nextInt();
        int b = s.nextInt();
        while (b == 0) {
            b = s.nextInt();
        }
        int c = a / b;
        System.out.println(c);
        s.close();
    }
}
```

Finally blok

Exception3.java

```
public class Exception3 {
    private static boolean simple() {
        try {
            int a = 10 / 0;
            System.out.println(a);
        } catch (ArithmeticException e) {
            System.out.println("Obrada greske... ");
            return false;
        } finally {
            System.out.println("Izvrsava se finally");
        }

        return true;
    }

    public static void main(String[] args) {
        System.out.println(simple());
    }
}
```

Problems @ Javadoc Declaration Console Call Hierarchy

```
<terminated> Exception3 [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javav
Obrada greske...
Izvrsava se finally
false
```

Upotreba finally bloka

- Osloboditi resurse i/ili reinicijalizovati promenljive desila se greška ili ne.

```
public static void printFile(String imeFajla) {
	BufferedReader br = null;
	try {
		br = new BufferedReader(new FileReader(imeFajla));
		String s;
		while ((s = br.readLine()) != null) {
			S.o.p(s);
		}
	} catch (FileNotFoundException fnfe) {
		S.o.p("Fajl " + imeFajla + " ne postoji");
	} catch (IOException ioe) {
		S.o.p("Greska prilikom citanja fajla " + imeFajla);
	} finally {
		if (br != null) {
			try {
				br.close();
			} catch (IOException ioe) {
				S.o.p("Greska kod zatvaranja fajla " + imeFajla);
			}
		}
	}
}
```

Pravljenje izuzetaka

- Možemo praviti naše izuzetke nasleđujući klasu Exception

```
public class MyException extends Exception {
	public MyException(String message) {
		super(message);
	}
}
```

Metoda može da generiše izuzetke

- Time se metodi koja ju je pozvala signalizira pojava greške
- Ključna reč **throw**

```
public void metod() throws MyException {
        ...
        if (nešto nije kako valja)
            throw new MyException("Nesto nije u redu");
}
```

Obraditi ili proslediti izuzetak?

- Metoda može ne obraditi checked izuzetak ali ga mora proslediti
  - To smo do sada imali na više primera

```
public RadnaOrganizacija(String imeFajla) throws IOException {
    ...
}
```

- Metoda A prosleđuje izuzetak metodi B koja je pozvala A samo ako B ume efektivnije i bolje da obradi grešku nego A

- Moguće je takođe i obraditi i proslediti izuzetak (rethrowing catched exception)

```
public void nekaMetoda() throws IOException {
    ...
    try {
        // neki kod koji može da izazove IOException
    } catch (IOException ioe) {
        System.out.println(ioe.getMessage());
        throw ioe;
    }
    ...
}
```

Izuzeci i petlje

Pretpostavimo da imamo sledeći kod

```
while (uslov) {
    blok1
    blok koji može prouzrokovati izuzetak
    blok2
}
```

Try-catch blok možemo postaviti na dva različita načina.

Izuzeci i petlje

```
while (uslov) {

    blok1

    try {
      // potencijalni izuzetak
    } catch (Izuzetak e) {
      // obrada greške
    }

    blok 2
}
```

Pojava izuzetka ne prekida petlju

```
try {
    while (uslov) {
      blok 1
      // potencijalni izuzetak
      blok 2
    }

} catch (Izuzetak e) {
    // obrada greške
}
```

Pojava izuzetka prekida petlju

Zadatak 1

- Putnički prtljag je predstavljen apstraktnom klasom Prtljag koja kao atribut ima težinu prtljaga.

- Klasa definiše konstruktor koji inicijalizuje sve atribute klase, get metode za sve atribute i apstraktnu metodu

  boolean izgubljen()

  koja kao rezultat vraća *true* ukoliko je prtljag zagubljen tokom transporta

- Neapstraktne klase RucniPrtljag i PredatiPrtljag nasleđuju klasu Prtljag.

- Rucni prtljag se ne može zagubiti, dok je verovatnoća da se zagubi predati prtljag 0.1

```java
public abstract class Prtljag {
	private int kilaza;

	public Prtljag(int kilaza) {
		if (kilaza < 0)
			throw new IllegalArgumentException("Negativna kilaza");

		this.kilaza = kilaza;
	}

	public int getKilaza() {
		return kilaza;
	}

	public abstract boolean izgubljen();
}
```

```java
public class RucniPrtljag extends Prtljag {
	public RucniPrtljag(int kilaza) {
		super(kilaza);
	}

	public boolean izgubljen() {
		return false;
	}
}
```

```java
public class PredatiPrtljag extends Prtljag {
	public PredatiPrtljag(int kilaza) {
		super(kilaza);
	}

	public boolean izgubljen() {
		return Math.random() <= 0.1;
	}
}
```

Zadatak 1

- Klasom Putnik je predstavljen jedan putnik na nekom avionskom letu. Svaki putnik ima ime, ručni prtljag i predati prtljag.

- Klasa definiše konstruktor koji inicijalizuje sve atribute klase i get metode za sve atribute klase.

- Klasom Let je predstavljen jedan avionski let. Kao atribute ova klasa ima naziv leta (String), nosivost aviona i niz putnika na letu. Klasa definiše konstruktor koji inicijalizuje sve atribute klase. **Ukoliko je ukupna težina putničkog prtljaga veća od nosivosti aviona konstruktor generiše izuetak VišakTereta.**

- Klasa Let takođe definiše metod int izgubljenPrtljag() koji kao rezultat vraća ukupnu težinu zagubljenog predatog prtljaga.

- Izuetak VišakTereta u konstruktoru prima naziv leta i formira poruku o grešci koja se prosleđuje konstruktoru nadklase.

```java
public class Putnik {
    private String ime;
    private RucniPrtljag rucni;
    private PredatiPrtljag predati;

    public Putnik(String ime, RucniPrtljag rucni, PredatiPrtljag predati) {
        this.ime = ime;
        this.rucni = rucni;
        this.predati = predati;
    }

    public String getIme() {
        return ime;
    }

    public RucniPrtljag getRucniPrtljag() {
        return rucni;
    }

    public PredatiPrtljag getPredatiPrtljag() {
        return predati;
    }
}
```

```java
public class VisakTereta extends Exception {
	public VisakTereta(String nazivLeta) {
		super("Visak tereta na letu " + nazivLeta);
	}
}

public class Let {
	private String nazivLeta;
	private int nosivost;
	private Putnik[] putnici;

	public Let(String nazivLeta, int nosivost, Putnik[] putnici)
		throws VisakTereta
	{
		this.nazivLeta = nazivLeta;
		this.nosivost = nosivost;
		this.putnici = putnici;

		int ukupnaTezina = 0;
		for (int i = 0; i < putnici.length; i++) {
			Putnik p = putnici[i];
			ukupnaTezina += p.getPredatiPrtljag().getKilaza();
			ukupnaTezina += p.getRucniPrtljag().getKilaza();
			if (ukupnaTezina > nosivost) {
				throw new VisakTereta(nazivLeta);
			}
		}
	}

	...
}
```

```
public int izgubljenPrtljag() {
	int ukupnaTezina = 0;

	for (int i = 0; i < putnici.length; i++) {
		Putnik p = putnici[i];
		PredatiPrtljag pp = p.getPredatiPrtljag();
		if (pp.izgubljen()) {
			ukupnaTezina += pp.getKilaza();
		}
	}

	return ukupnaTezina;
}

public String toString() {
	return nazivLeta + ", " + nosivost;
}
```

Zadatak 1

- Klasom SpisakPutnika je predstavljen neki spisak (niz) putnika.

- Informacije o putnicima se čitaju iz ulaznog fajla koji je formatiran na sledeći način
  - Prva linija fajla – broj putnika
  - Svaka sledeća linija nosi informacije o jednom putniku i to ime, težinu ručnog prtljaga i težinu predatog prtljaga

- **Ne pretpostavljamo da je ulazni fajl dobro formatiran i obrađujemo sve greške u radu sa ulaznim fajlom.**

Primer ulaznog fajla

7
mika, 8, 20
zika, 7, 21
pera, 5, 18
ana, 9, 23
zivana, 10, 15
mina, 0, 10
stojan, 5, 25

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class SpisakPutnika {
    private Putnik[] putnici;

    public boolean ucitajSpisak(String putniciFajl) {..

    public Putnik[] getPutnici() {
        return putnici;
    }
}
```

```
public boolean ucitajSpisak(String putniciFajl) {
	BufferedReader br = null;
	try {
		br = new BufferedReader(new FileReader(putniciFajl));
		int brojPutnika = Integer.parseInt(br.readLine());
		putnici = new Putnik[brojPutnika];
		for (int i = 0; i < brojPutnika; i++) {
			String linija = br.readLine();
			String[] tok = linija.split(",");
			if (tok.length == 3) {
				String ime = tok[0].trim();
				int tezinaRucni = Integer.parseInt(tok[1].trim());
				int tezinaPredati = Integer.parseInt(tok[2].trim());
				putnici[i] = new Putnik(ime,
					new RucniPrtljag(tezinaRucni),
					new PredatiPrtljag(tezinaPredati));
			} else {
				S.o.p("Fajl " + putniciFajl +
					" nije ispravno formatiran");
				return false;
			}
		}
		return true;
	} catch (NumberFormatException nfe) {
		S.o.p(nfe.getMessage()); return false;
	} catch (IllegalArgumentException iae) {
		S.o.p(iae.getMessage()); return false;
	} catch (IOException ioe) {
		S.o.p(ioe.getMessage()); return false;
	} finally {
		if (br != null) {
			try { br.close();
			} catch (IOException e) { System.out.println("..."); }
		}
	}
}
```

Zadatak 1

- Klasa SimulacijaLeta definiše main metod u kome se kreira spisak putnika iz nekog fajla, te se potom kreira let proizvoljne nosivosti koji treba da preveze putnike sa spiska. Na kraju metoda se štampa ukupna težina izgubljenog prtljaga.

```
public class SimulacijaLeta {
	public static void main(String[] args) {
		SpisakPutnika sp = new SpisakPutnika();
		boolean ok = sp.ucitajSpisak("putnici.txt");
		if (ok) {
			try {
				Putnik[] p = sp.getPutnici();
				Let l = new Let("JU113", 200, p);
				S.o.p("Tezina izgubljenog prtljaga: " +
					l.izgubljenPrtljag());
			} catch (VisakTereta e) {
				S.o.p(e.getMessage());
			}
		}
	}
}
```