Objektno orijentisano programiranje 2

Grafički korisnički interfejs

Uvod

- Grafički korisnički interfejs je karakteristika gotovo svih savremenih aplikacija
- Termin na engleskom: *Graphical User Interface* (GUI)
- GUI je vezan za rad sa prozorima:
  - izlazni podaci se prikazuju u prozorima
  - ulazni podaci generišu događaje u prozorima
- Aplikacije koje imaju samo tekstualni ulaz i izlaz
  - nazivaju se konzolnim aplikacijama
  - upravljaju celim ekranom (odnosno prozorom koji simulira ceo ekran konzole)
- Aplikacija sa GUI ne upravlja celim ekranom, već prozorima koje kreira
  - izuzetno može upravljati i celim ekranom (*full screen mode*)
- Java je prvi široko rasprostranjeni programski jezik koji na *de facto* standardan način podržava programiranje GUI

Paket AWT

- Podrška za programiranje GUI nalazi se u `java.awt paketu`
- Noviji paketi/bibliotekeza programiranje GUI: `javax.swing`, `javafx`
- AWT je skraćenica od *Abstract Windowing Toolkit*
  - apstraktni alati za rad sa prozorima
  - apstraktni: ne zavise od konkretne platforme
- `java.awt` paket sadrži klase i interfejse koji podržavaju izlazne i ulazne aspekte GUI
- `java.awt` paket se koristi za programiranje
  - samostalnih aplikacija
  - apleta
- Komponente koje se pojavljuju na ekranu nazivaju se i "kontrole" ili "vidžiti"
  - primeri: ekranski tasteri (*button*), radio-dugmad (*radio-button*), polja za potvrde (*checkbox*), klizači (*scrollbar*), polja za tekst (*text box*), liste (*list*), padajuće liste (*combo-box, choice*)

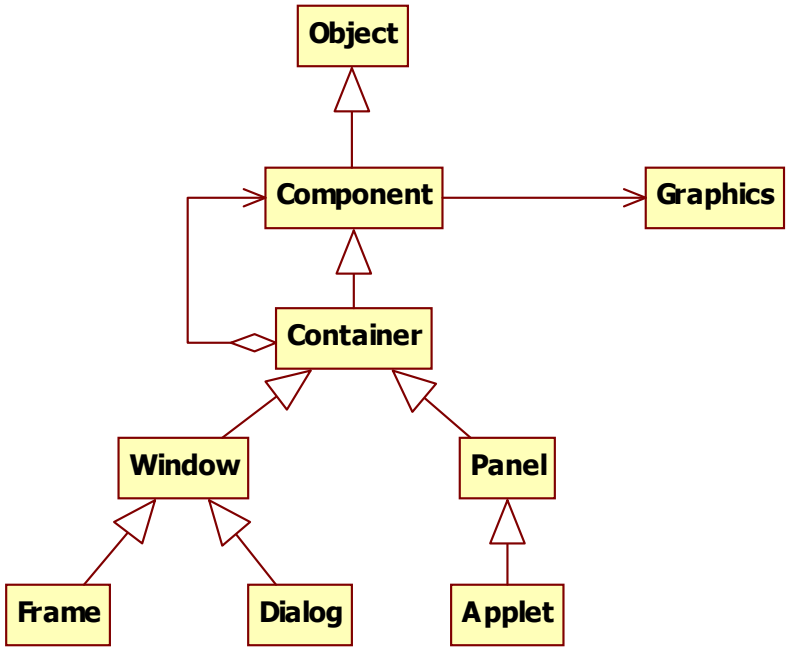
Važnije klase iz paketa `java.awt`

- Klasa `Component` je zajednička osnovna klasa za sve GUI kontrole
- Klasa `Component` reprezentuje nešto što:
  - ima poziciju i veličinu može se iscrtati na ekranu i prihvata ulazne događaje
- Klasa `Container` je izvedena iz `Component`
  - objekat `Container` može da sadrži druge AWT komponente
- Klasa `Window` je izvedena iz `Container` (sadrži komponente)
  - njeni objekti su prozori najvišeg nivoa – nisu sadržani u drugim komponentama
  - obuhvata metode za rad sa prozorima
- Klasa `Frame` je izvedena iz `Window` – glavni prozori aplikacije
  - ima naslovnu traku sa kontrolnim dugmadima i okvir
  - može da sadrži traku menija
- Klasa `Dialog` je izvedena iz `Window` – prozori dijaloga
  - dijalog ima roditeljski prozor
    - dijalog nestaje sa ekrana kad se roditelj minimizuje
  - može da bude modalan ili ne
  - ne sadrži traku menija

Grafički kontekst

- Grafički kontekst sadrži skup atributa koji određuju način crtanja/pisanja
  - atributi – promenljive stanja za operacije crtanja i pisanja po komponenti
  - crtanje i pisanje se obavlja tekućim vrednostima atributa konteksta
    - boja
    - font
    - odsecajući region
    - ...
  - grafičke operacije modifikuju bite samo unutar odsecajućeg (*clipping*) regiona
- Klasa `Graphics` omogućava crtanje/pisanje na komponentama
  - apstraktni grafički kontekst komponente na ekranu (ili van ekrana, samo u memoriji)
  - crtanje i pisanje se obavlja nad objektom klase (izvedene iz) `Graphics`
  - dohvatanje objekta `Graphics` odgovarajućeg objekta `Component`
    - pomoću metoda `getGraphics()`
  - metoda komponente `paint()` dobija referencu na objekat tipa `Graphics`
    - metodu ne poziva program, već AWT nit

Hijerarhija važnijh klasa paketa AWT



Događaji

- Aplikacije sa GUI su "vođene događajima"
  - programska paradigma – prirodno se povezuje sa OO paradigmom
- Centralizovana obrada događaja – klasa `Event`
  - definiše sve događaje u obliku klasnih (statičkih) celobrojnih konstanti
  - konstante se koriste u obradi događaja da se prepozna vrsta događaja
  - atribut objekta događaja `id` određuje vrstu događaja
    - vrednost se poredi sa konstantama
  - obezbeđuje i metode za određivanje
    da li su za vreme događaja pritisnuti tasteri-modifikatori
    - <Ctrl>, <Shift>, <Alt>
  - zastareo koncept – nije u duhu OO
- Noviji koncept – delegirana obrada događaja, u duhu OO

Primer "Zdravo“ – Java 1.0

```
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);
      return true;
    } else return false;
  }
  public static void main(String args[]){
    Prozor prozor = new Prozor(); }
}
```

O primeru “Zdravo” (1)

- Klasa `Prozor` realizuje jednostavnu aplikaciju sa GUI
- Klasa `Prozor` se izvodi iz klase `Frame` i sadrži:
  - konstruktor
  - metode: `paint`, `handleEvent` i `main`
- Konstruktor
  - poziva konstruktor klase `Frame` prosleđujući mu naslov
  - poziva metod `setSize()` klase `Component` koji određuje dimenzije prozora
  - poziva metod `setVisible()` klase `Window` koji prikaže prozor
- 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
  - ovde `handleEvent()` metod detektuje zatvaranje prozora aplikacije kao `WINDOW_DESTROY` događaj

O primeru “Zdravo” (2)

- Metod `paint()`
  - metod klase `Component` koji se automatski poziva iz izvršnog okruženja (iz AWT niti) da se iscta ta komponenta
  - metod se preklapa u izvedenim klasama da definiše kako će se iscrtati objekat (ovde prozor aplikacije)
  - poziva se za inicijalno iscrtavanje
    kao i prilikom pomeranja, promene veličine i pokrivanja pa otkrivanja prozora
  - kao parametar mu se prosleđuje objekat tipa `Graphics`
    - ovaj objekat se koristi da se ažurira prikaz komponente crtanjem ili pisanjem na njenoj grafičkoj površi
  - tačka (0,0) koordinatnog sistema grafičkog prikaza se poklapa sa gornjim levim uglom komponente
  - odsecajući region grafičkog prikaza je granični pravougaonik oko komponente
  - ovde `paint()` metod ispisuje tekst `"Zdravo!"` u prozoru aplikacije, počevši od tačke (50,50)
- Metod `main()` kreira objekat tipa `Prozor`
  - startuje se AWT nit grafičkog okruženja
  - main po izvršenju tela čeka na završetak AWT niti

Novi koncept obrade događaja

- Java 1.1 uvodi novi koncept događaja – delegiranu obradu
- Događaje generišu izvori (*sources*) događaja
- Događaje obrađuju objekti klasa osluškivača (*listeners*) događaja
- Klase osluškivača implementiraju interfejs nekog osluškivača događaja
- Jedan ili više objekata osluškivača se može registrovati kod nekog izvora
  - registrovani osluškivači će biti obaveštavani od izvora o događaju
  - automatski im se akivira obrada događaja pojedine vrste
- Metodi obrade događaja (rukovaoci - *handlers*)
  - "propisani" su interfejsom odgovarajućeg osluškivača
- Ovakav model obrade događaja se naziva "delegiranje" ili "prosleđivanje":
  - sposobnost obrade događaja se delegira svakom objektu koji implementira interfejs odgovarajućeg osluškivača
- U Javi 1.0 obrada događaja je centralizovana
  - centralni metod (`handleEvent`) obrade za sve događaje
  - obrada događaja u razgranatoj kontrolnoj strukturi - neobjektno

Klase događaja

- U `java.util`:
  - klasa `EventObject`
    - koren hijerarhije događaja prema modelu delegirane obrade
    - metod `Object getSource();`
- U `java.awt`:
  - klasa `AWTEvent`
    - potklasa klase `EventObject`
    - natklasa svih AWT događaja koji koriste model delegirane obrade
    - parametri konstruktora: objekat izvora i celobrojna vrsta događaja
    - metod `int getID();`
      - vraća ceo broj koji opisuje vrstu događaja
- U `java.awt.event`:
  - razne klase događaja izvedene iz `AWTEvent`

Osluškivači događaja

- Program koji obrađuje događaje ima karakteristične delove koda
- U zaglavlju klase osluškivača
  - klasa implemetira interfejs nekog osluškivača
    - ili klasa proširuje neku klasu koja implementira interfejs osluškivača
  - na primer:
    ```
    public class Obradjivac implements ActionListener {
    ```
- U telu klase osluškivača
  - implementacija metoda rukovalaca koje propisuje interfejs osluškivača
  - na primer:
    ```
    public void actionPerformed(ActionEvent e) {/*...*/}
    ```
- Registrovanje objekta klase osluškivača događaja kod izvora
  - na primer:
    ```
    komponentaIzvor.addActionListener(obradjivac);
    ```

Primer delegirane obrade događaja

```
import java.awt.*; import java.awt.event.*;
class BrojanjeDogadjaja extends Frame implements ActionListener {
   private Button dugme;   private int broj=0;
   public BrojanjeDogadjaja(){
        super("Brojanje"); 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) {
        new BrojanjeDogadjaja(); }
}
```

Primer delegirane obrade - izlaz

Klase adaptera

- Problem
  - u klasi konkretnog osluškivača se moraju definisati svi metodi interfejsa
  - neki metodi u klasi konkretnog osluškivača ostaće prazni
  - kod postaje glomazan, nečitak i težak za održavanje
- AWT definiše klase adaptera da reši gornji problem
  - za sve osluškivače sa više od jednog metoda postoji klasa adaptera
  - adapter implementira sve metode interfejsa osluškivača kao prazne
- Korišćenje klasa adaptera
  - iz odgovarajućeg adaptera se izvodi klasa željenog osluškivača
  - izvedena klasa treba da redefinise samo željene obrade događaja
  - umesto implementacije interfejsa radi se proširivanje adaptera
- Nedostatak
  - nije moguće istovremeno izvesti klasu iz adaptera i iz neke druge klase

Primer proširivanja adaptera

```
import java.awt.*;
import java.awt.event.*;
public class PrimerAdaptera extends Frame {
  class AdapterProzora extends WindowAdapter{
    public void windowClosing(WindowEvent we){dispose();}
  };
  public PrimerAdaptera() {
    super("Primer adaptera"); setSize(280,80);
    addWindowListener(new AdapterProzora()); setVisible(true);
  }
  public void paint(Graphics g) {g.drawString("Zdravo!",50,50);}
  public static void main(String args[]){
    new PrimerAdaptera();
  }
}
```

Isti primer – anonimna klasa

```
import java.awt.*;
import java.awt.event.*;
public class PrimerAnonimnogAdaptera extends Frame {
  public PrimerAnonimnogAdaptera() {
    super("Anonimni adapter"); setSize(280,80);

    addWindowListener(new WindowAdapter(){
      public void windowClosing(WindowEvent we){dispose();}
    });

    setVisible(true);
  }
  // ...
}
```

Obrada događaja koji potiču od miša

- Postoje dva interfejsa za osluškivanje događaja koji potiču od miša
  - `MouseListener`: osluškivač miša i
  - `MouseMotionListener`: osluškivač kretanja miša
- Interfejs osluškivača miša predviđa 5 metoda:

```
public void mouseEntered (MouseEvent d); //kurzor usao u polje komp.
public void mouseExited  (MouseEvent d); //kurzor izasao iz polja komp.
public void mousePressed (MouseEvent d); //pritisnuto dugme
public void mouseReleased(MouseEvent d); //otpusteno dugme
public void mouseClicked (MouseEvent d); //pritisnuto i otpusteno dugme
```

  - ako se dugme otpusti na istoj poziciji gde je pritisnuto – `mouseClicked`
  - ako se dugme otpusti na različitoj poziciji – `mouseReleased`
- Interfejs osluškivača kretanja miša predviđa 2 metoda:

```
public void mouseMoved   (MouseEvent d); //pomeren bez pritiskanja dug.
public void mouseDragged (MouseEvent d); //pomeren sa pritisnutim dug.
```

Klasa `MouseEvent`

- Klasa `MouseEvent` je iz paketa `java.awt.event`
  - u hijerarhiji klasa na sledećoj poziciji:
    `java.awt.event.MouseEvent->java.awt.event.InputEvent->`
    `java.awt.event.ComponentEvent->java.awt.AWTEvent->`
    `java.util.EventObject->java.lang.Object`
- Konstruktor:
  `public MouseEvent(Component` *`source`*`, int` *`id`*`, long` *`when`*`, int` *`modifiers`*`,`
  `int` *`x`*`, int` *`y`*`, int` *`clickCount`*`, boolean` *`popupTrigger`*`)`
  - `source`: komponenta koja je izazvala događaj
  - `id`: tip događaja (npr. `MOUSE_CLICKED`, `MOUSE_DRAGGED`,...)
  - `when`: *timestamp* trenutka kada se događaj desio
  - `modifiers`: modifikatori koji određuju da li je pritisnut <ALT>, <SHIFT>, <MOUSE_BUTTONx>
  - `x,y`: koordinate tačke gde je se nalazio pointer pri događaju
  - `clickCount`: broj "klikova" kojima je izazvan događaj
  - `popupTrigger`: informacija da li je događaj izazvao pojavu *pop-up* menija

Primer osluškivača miša

```
import java.awt.*;
import java.awt.event.*;
public class OsluskivacMisa extends Frame implements MouseListener {
  private String t="Ceka se na dogadjaj od 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="Dog: clicked";repaint();}
  public void mouseEntered (MouseEvent d){t="Dog: entered";repaint();}
  public void mouseExited  (MouseEvent d){t="Dog: exited"; repaint();}
  public void mousePressed (MouseEvent d){t="Dog: pressed";repaint();}
  public void mouseReleased(MouseEvent d){t="Dog: released";repaint();}
  public static void main(String[] args){ new OsluskivacMisa(); }
}
```

Primer osluškivača kretanja miša

```
import java.awt.*;
import java.awt.event.*;
class OsluskivacKretanjaMisa extends Frame implements MouseMotionListener {
  private String t="Ceka se dogadjaj kretanja misa...";
  public OsluskivacKretanjaMisa(){
   super("Osluskivac kretanja misa"); setSize(400,100);
   addMouseMotionListener(this); setVisible(true);
  }
  public void paint(Graphics g){ g.drawString(t,50,50); }
  public void mouseMoved (MouseEvent d) {
   t="Dog: mouseMoved ("+d.getX()+","+d.getY()+")"; repaint();
  }
  public void mouseDragged (MouseEvent d) {
   t="Dog: mouseDragged ("+d.getX()+","+d.getY()+")"; repaint();
  }

  public static void main(String[] args){ new OsluskivacKretanjaMisa(); }
}
```

Standardni AWT osluškivači (1)

Interfejs	Adapter	Metodi
`ActionListener`	*nema*	`actionPerformed`
`AdjustmentListener`	*nema*	`adjustmentValueChanged`

`ComponentListener`	`ComponentAdapter`	`componentShown`
		`componentHidden`
		`componentMoved`
		`componentResized`

`ContainerListener`	`ContainerAdapter`	`componentAdded` `componentRemoved`
`FocusListener`	`FocusAdapter`	`focusGained` `focusLost`
`ItemListener`	*nema*	`itemStateChanged`

`KeyListener`	`KeyAdapter`	`keyPressed`
		`keyReleased`
		`keyTyped`

Standardni AWT osluškivači (2)

`MouseListener`	`MouseAdapter`	`mouseEntered`
		`mouseExited`
		`mousePressed`
		`mouseClicked`
		`mouseReleased`

`MouseMotionListener`	`MouseMotionAdapter`	`mouseDragged` `mouseMoved`
`TextListener`	*nema*	`textValueChanged`

`WindowListener`	`WindowAdapter`	`windowOpened`
		`windowClosing`
		`windowClosed`
		`windowActivated`
		`windowDeactivated`
		`windowIconified`
		`windowDeiconified`

Grupe AWT događaja (1)

- AWT događaji se mogu podeliti u 2 grupe:
  - događaje niskog nivoa
    - reprezentuju elementarna zbivanja u sistemu prozora ili elementarne ulaze
  - semantičke događaje
    - rezultat su korisničkih akcija koje su specifične za komponente
- Događaji koje generišu komponente, kontejneri, fokusi i prozori
  - događaji su niskog nivoa
  - 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 tastature
      (fokus tastature je sposobnost da se prihvate karakteri koji se unose preko tastature)
  - događaji prozora se generišu
    - da daju informaciju o stanju prozora

Grupe AWT događaja (2)

- Elementarni događaji koji potiču od ulaza miša ili tastature
  - događaji su niskog nivoa
  - događaji koji potiču od miša su podeljeni u dve grupe:
    - događaje miša: klik, pritisnut/otpušten taster, ušao/izašao iz komponente
    - događaje kretanja miša: pomeranje i prevlačenje
  - događaji kretanja miša su češći
    - za njih je definisan poseban interfejs, da se ne bi morali uvek obrađivati
- Semantički događaji uključuju događaje akcije, članske, tekstualne i prilagođenja
  - događaje akcije generišu
    - ekranski tasteri (pritisak), stavke menija, liste i tekst polja
  - članske događaje generiše
    - izbor jedne od stavki iz liste, padajuće liste ili polja za potvrdu (dugmeta grupe radio-dugmadi)
  - tekstualni događaji se generišu
    - kada se menja tekst u prostoru za tekst ili u polju za tekst
  - događaji prilagođenja se generišu
    - kada korisnik promeni vrednost klizača (*scrollbar*)

Događaji koje generišu komponente (1)

AWT komponenta	Tipovi događaja koje komponenta može da generiše										
	action	adjust-ment	com-ponent	con-tainer	focus	item	key	mouse	mouse motion	text	window
`Button`	X		X		X		X	X	X		
`Canvas`			X		X		X	X	X		
`Checkbox`			X		X	X	X	X	X		
`CheckboxMenuItem`	*					X					
`Choice`			X		X	X	X	X	X		
`Component`			X		X		X	X	X		
`Container`			X	X	X		X	X	X		

* `CheckboxMenuItem` nasleđuje `addActionListener` od `MenuItem`, ali ne generiše događaje akcije

Događaji koje generišu komponente (2)

AWT komponenta	Tipovi događaja koje komponenta može da generiše										
	action	adjust-ment	com-ponent	con-tainer	focus	item	key	mouse	mouse motion	text	window
`Dialog`			X	X	X		X	X	X		X
`Frame`			X	X	X		X	X	X		X
`Label`			X		X		X	X	X		
`List`	X		X		X	X	X	X	X		
`MenuItem`	X										
`Panel`			X	X	X		X	X	X		

Događaji koje generišu komponente (3)

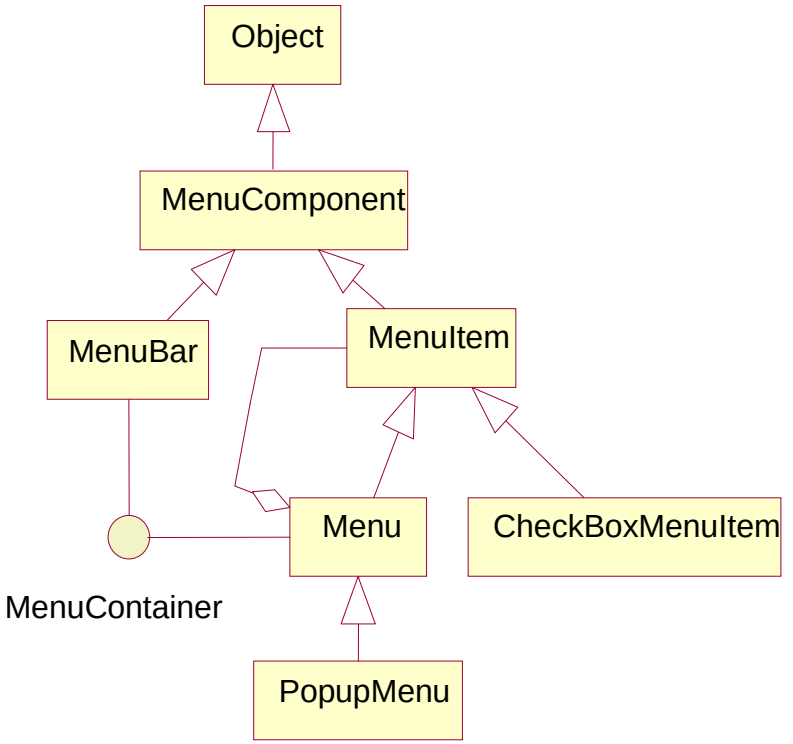
AWT komponenta	Tipovi događaja koje komponenta može da generiše										
	action	adjust-ment	com-ponent	con-tainer	focus	item	key	mouse	mouse motion	text	window
`Scrollbar`		X	X		X		X	X	X		
`ScrollPane`			X	X	X		X	X	X		
`TextArea`			X		X		X	X	X	X	
`TextComponent`			X		X		X	X	X	X	
`TextField`	X		X		X		X	X	X	X	
`Window`			X	X	X		X	X	X		X

Meniji – klase i interfejsi (1)

- Klasa `MenuComponent` je bazna klasa koja sadrži metode za rad sa menijima
- Klasa `MenuBar` implementira traku menija (pridružuje se prozoru aplikacije)
  - klasa `MenuBar` se izvodi iz klase `MenuComponent`
  - objekat klase `MenuBar` se pridružuje objektu klase `Frame`
    - metodom `setMenuBar()` klase `Frame`
- Klasa `MenuItem` implementira pojedinačne stavke menija
  - klasa `MenuItem` se izvodi iz klase `MenuComponent`
  - sadrži metod za postavljanje stringa komande `setActionCommand(String)`, kao i
    - postavljanje i dohvatanje labela: `setLabel(String)`, `String getLabel()`
    - o(ne)mogućavanje: `setEnabled(boolean)`
    - postavljanje prečica: `setShortcut(MenuShortcut)`
- Klasa `Menu` implementira padajuće menije
  - izvedena je iz klase `MenuItem`
  - objekat klase `Menu` može sadržati druge `MenuItem` objekte i tako formirati kaskadne menije
  - klasa `Menu` sadrži metode za dodavanje objekata klase `MenuItem` i separatora u objekte klase `Menu`
  - klasa `Menu` sadrži i metode za pristup objektima `MenuItem` unutar objekta `Menu`
  - objekat klase `MenuBar` sadrži jedan ili više objekata klase `Menu`

Meniji – klase i interfejsi (2)

- Klasa `CheckboxMenuItem` implementira stavke koje mogu biti obeležene potvrdom
  - klasa `CheckboxMenuItem` se izvodi iz klase `MenuItem`
  - sadrži metode koje postavljaju znak potvrde i očitavaju status potvrđenosti
- Klasa `PopupMenu` implementira "iskačuće" menije
  - izvodi se iz klase `Menu`
  - pojavljuju se na zadatoj poziciji u prostoru iznad odgovarajuće komponente
- Interfejs `MenuContainer` implementiraju klase koje sadrže objekte menija
  - klase `Frame`, `Menu`, `MenuBar` i druge
  - metoda: `void remove(MenuComponent mk)`



Kreiranje menija i obrada događaja

- Meni aplikacije se kreira tako što se:
  - kreira objekat `MenuBar` i kreiraju objekti `Menu`
  - dodaju objekti `MenuItem` objektu klase `Menu`
    - pozivom metoda *`meni`*`.add(`*`naziv`*`)` ili *`meni`*`.add(MenuItem)`
  - dodaju objekti `Menu` u objekat `MenuBar`
    - pozivom metoda *`meniTraka`*`.add(`*`meni`*`)`
  - postavi meni prozora aplikacije pozivom *`prozor`*`.setMenuBar(`*`meniTraka`*`)`
- Tehnika delegirane obrade događaja iz menija
  - zasniva se na interfejsu `ActionListener`
  - klasa koja osluškuje događaje iz menija treba da implementira interfejs `ActionListener`
  - objekat te klase se registruje kao slušalac objekta određenog menija
    `meni``.addActionListener(`*`osluskivac`*`);`
  - piše se metod `public void actionPerformed (ActionEvent e)`
  - u metodi `actionPerformed` ime aktivirane stavke menija se dohvata
    `e.getActionCommand()`

Primer menija (1)

```
import java.awt.*;
import java.awt.event.*;
public class PrimerMenija extends Frame
                            implements ActionListener {
  String izborIzMenija = "Izaberite stavku iz menija...";
  public PrimerMenija() {
    super("Meni");
    setSize(300,200);
    dodajMenije();
    setVisible(true);
  }
```

Primer menija (2)

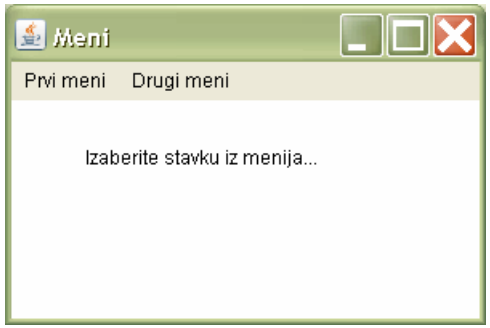
```
void dodajMenije() {
  MenuBar trakaMenija = new MenuBar();
  Menu prviMeni = new Menu("Prvi meni");
  Menu drugiMeni = new Menu("Drugi meni");
  prviMeni.add("Prvi meni, prva stavka");
  prviMeni.add("Prvi meni, druga stavka");
  prviMeni.add("Kraj");
  prviMeni.addActionListener(this);
  drugiMeni.add("Drugi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

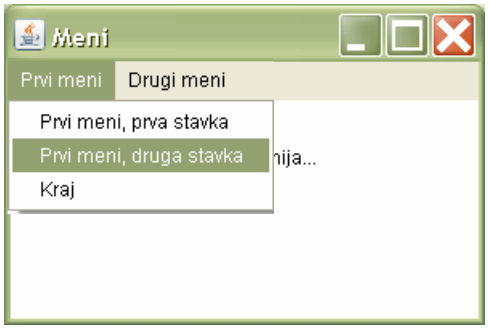
Primer menija (3)

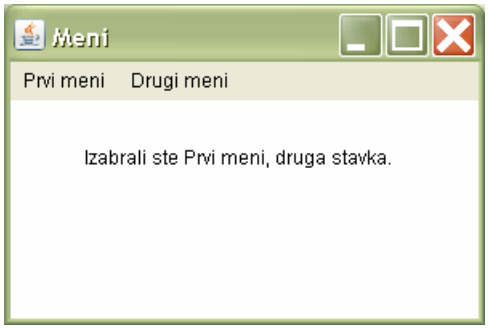
```
  public void paint(Graphics g) {
    g.drawString(izborIzMenija,50,100);
  }
  public void actionPerformed (ActionEvent e) {
    String komanda=e.getActionCommand();
    if(komanda.equals("Kraj")) dispose();
    else{
      izborIzMenija = "Izabrali ste "+komanda+".";
      repaint();
    }
  }
  public static void main(String args[]){
    PrimerMenija prozor = new PrimerMenija();
  }
}
```

Primer menija (4)

- Izlaz







- Metod `repaint()` se koristi da se prozor ponovo iscrta
  - ovaj metod izaziva poziv metoda `paint()`
  - metod `paint()` se automatski poziva kada se koriste metodi: `setVisible(true)`, `show()`-zastareo, `repaint()` ili `update()`

Kreiranje dijaloga

- Klasa `Dialog` implementira prozore dijaloga kroz koje se komunicira sa korisnikom
- Dva tipa dijaloga se mogu kreirati:
  - modalni dijalozi
    - dok su otvoreni, fokus se ne može preneti na druge prozore aplikacije
  - nemodalni dijalozi
    - fokus se može preneti i na druge prozore aplikacije dok su otvoreni
- Dijalog se kreira sledećim konstruktorom:

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

- Roditelj je glavni prozor aplikacije
- Nakon kreiranja dijaloga, ovaj se može otvarati i zatvarati sa

```
setVisible(Boolean)
```

Primer dijaloga (1)

```
import java.awt.*;
import java.awt.event.*;
public class PrimerDijaloga extends Frame implements ActionListener{
  class Dijalog extends Dialog {
    Dijalog(Frame roditelj) {
      super(roditelj,"Dijalog",false); setSize(130,80);
      addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent we){setVisible(false);}
      });
    }
  }
  private Dijalog dijalog;
  public PrimerDijaloga() {
    super("Primer dijaloga"); setSize(130,80);
    dodajMenije(); dijalog = new Dijalog(this); setVisible(true);
  }
```

Primer dijaloga (2)

```
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("Kraj")) dispose();
    else if (komanda.equals("Otvori")) dijalog.setVisible(true);
    else if (komanda.equals("Zatvori")) dijalog.setVisible(false);
  }
  public static void main(String args[]){
    PrimerDijaloga prozor = new PrimerDijaloga();
  }
}
```

Panel

- Klasa `Panel` služi za organizovanje komponenti u prozoru
  - izvedena je iz klase `Container`
  - predstavlja najjednostavniju kontejnersku komponentu
  - predstavlja prostor u koji se mogu smeštati druge komponente
  - komponente koje se smeštaju na panele uključuju i druge panele
  - prima događaje prouzrokovane:
    - mišem,
    - tastaturom i
    - promenom fokusa
- Paneli se definišu, postavlja im se pozadina, dodaju im se komponente (npr. ekranski tasteri)

Primer panela (1)

```
import java.awt.*;
import java.awt.event.*;
public class PrimerPanela extends Frame {
  public PrimerPanela() {
    super("Panel primer");
    setSize(150,100);
    dodajPanele();
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e){
        dispose();
      }
    });
    setVisible(true);
  }
```

Primer panela (2)

```
 void dodajPanele() {
    Panel panel1 = new Panel();
    panel1.setBackground(Color.green);
    panel1.add(new Button("jedan"));
    panel1.add(new Button("dva"));
    add(panel1, "West");
    Panel panel2 = new Panel();
    panel2.setBackground(Color.yellow);
    panel2.add(new Button("tri"));
    panel2.add(new Button("cetiri"));
    add(panel2, "South");
  }
  public static void main(String args[]){
    PrimerPanela prozor = new PrimerPanela();
  }
}
```

Rasporedi (planovi)

- Komponente u kontejneru raspoređuje odgovarajući upravljač rasporeda/plana (*layout manager*)
- Svaka klasa upravljača rasporeda mora implementirati interfejs
  - `LayoutManager`
- Postoje sledeće klase upravljača rasporeda:
  - `FloatLayout`
    - raspoređuje komponente sleva-udesno u niz potrebnih redova
  - `BorderLayout`
    - raspoređuje komponente po ivicama i u sredinu kontejnera
  - `CardLayout`
    - raspoređuje komponente kao karte u špilu karata (jedna iza druge)
  - `GridLayout`
    - raspoređuje komponente u pravilnoj rešetki
  - `GridBagLayout`
    - raspoređuje komponente prema skupu objekata `GridBagConstraints`
- Za panele je podrazumevan `FloatLayout`, a za prozore `BorderLayout`

Primer rasporeda (1)

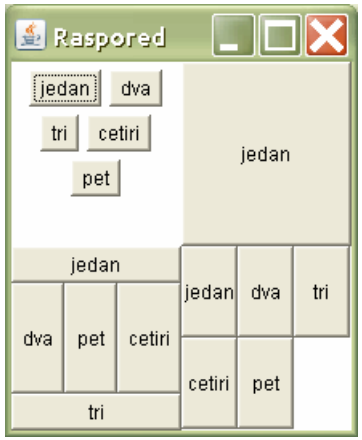
```
import java.awt.*;
import java.awt.event.*;
public class PrimerRasporeda extends Frame {
  public PrimerRasporeda() {
    super("Raspored");
    dodajPanele();
    setSize(220,270);
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e){ dispose(); }
    });
    setVisible(true);
  }
```

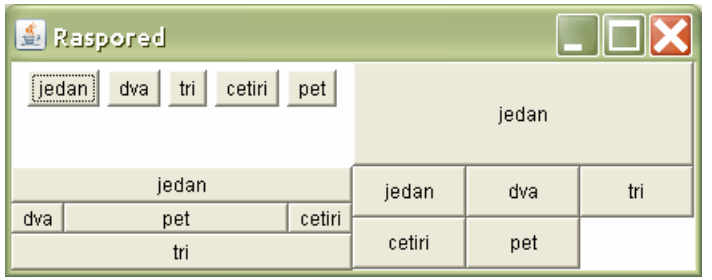
Primer rasporeda (2)

```
void dodajPanele() {
   setLayout(new GridLayout(2,2));
   Panel flow = new Panel();
   Panel card = new Panel();
   Panel border = new Panel();
   Panel grid = new Panel();
   card.setLayout(new CardLayout());
   border.setLayout(new BorderLayout());
   grid.setLayout(new GridLayout(2,3));
   dodajTastere(flow); dodajTastere(card);
   dodajTastere(border); dodajTastere(grid);
   add(flow); add(card); add(border); add(grid);
 }
```

Primer rasporeda (3)

```
 void dodajTastere(Panel panel){
    panel.add(new Button("jedan"), "North");
    panel.add(new Button("dva"), "West");
    panel.add(new Button("tri"), "South");
    panel.add(new Button("cetiri"), "East");
    panel.add(new Button("pet"), "Center");
  }
  public static void main(String args[]){
    PrimerRasporeda prozor = new PrimerRasporeda();
  }
}
```





Opštenamenske GUI komponente

- Klasa `Component` (dete klase `Object`) je roditeljska klasa za GUI kontrole
- U opštenamenske GUI komponente (kontrole) se ubrajaju klase:
  - Label - natpis, labela (statički tekst)
  - Button - taster
  - Checkbox - polje za potvrdu ili radio-dugme (ako se pridruži klasi `CheckboxGroup`)
  - List - lista
  - Choice - padajuća lista (*combo-box*)
  - TextField - tekstualno polje (dinamički tekst)
  - TextArea - prostor za višelinijski tekst
  - ScrollBar - klizač
  - ScrollPane - panel sa klizačima za pomeranje sadržane komponente
  - Canvas - kanvas (pravougaoni prostor na ekranu po kojem se može crtati)
- Klasa Component sadrži više od 70 metoda
  - metodi su zajednički za razne komponenete (kontrole)

Natpisi i tasteri

- Klasa `Label` se koristi za prikazivanje teksta koji se može samo čitati
  - tekst labele se ne može menjati kroz korisnički interfejs, ali može programski
- Obezbeđuje metode:
  - za upis i čitanje teksta i
  - Za poravnanje teksta unutar objekta natpisa
- Klasa `Button` se koristi za ekranske tastere
- Tasteri su označeni natpisom, slike na tasterima nisu podržane
- Decentralizovana obrada događaja tastera:
  - pritisak tastera (*click*) rezultuje u događaju tipa `ActionEvent`
  - ovaj događaj se obrađuje metodom `actionPerformed(ActionEvent e)`
  - izvor događaja `e.getSource()` je objekat tipa `Button` koji je izazvao događaj
  - argument događaja može da vrati tekst natpisa tastera koji je pritisnut `e.getActionCommand()`

Primer natpisa i tastera (1)

```
import java.awt.*;
import java.awt.event.*;
public class PrimerTastera extends Frame implements ActionListener {
  Label labela = new Label("Pocetni tekst");
  public PrimerTastera() {super("Tasteri");
    dodajKomponente(); setSize(250,100); setVisible(true);
  }
  void dodajKomponente() {
    add(labela, "North"); labela.setAlignment(Label.CENTER);
    Button jedan=new Button("jedan"),
           dva=new Button("dva"), tri=new Button("tri");
    tri.setActionCommand("Kraj");
    jedan.addActionListener(this);
    dva.addActionListener(this);
    tri.addActionListener(this);
    Panel panel = new Panel();
    panel.add(jedan); panel.add(dva); panel.add(tri);
    add(panel, "Center");
  }
```

Primer natpisa i tastera (2)

```
  public void actionPerformed(ActionEvent e) {
    labela.setText(e.getActionCommand());
    if(labela.getText().equals("Kraj"))
      dispose();
  }
  public static void main(String args[]){
    PrimerTastera prozor = new PrimerTastera();
  }
}
```

Polja za potvrde i radio-dugmad

- Klasa `Checkbox` omogućava kreiranje
  - polja za potvrdu i
  - radio-dugmadi
- Objektu polja za potvrdu je pridružena jedna labela, a stanje objekta je tipa boolean
- Klasa sadrži metode za dohvatanje i modifikovanje stanja i labele
- Promena stanja prouzrokuje `ItemEvent` događaj
  - događaj se obrađuje metodom `itemStateChanged()`
- Klasa `CheckboxGroup` se koristi za grupisanje `Checkbox` objekata
- Grupisani `Checkbox` objekti se ponašaju kao radio-dugmad
- Ako nije u grupi, objekat se ponaša kao obično polje za potvrdu

Primer polja za potvrdu i radio-dugmadi (1)

```
import java.awt.*; import java.awt.event.*;
public class PrimerPoljaZaPotvrdu extends Frame
                                          implements ItemListener {
  Label labela = new Label("Pocetni tekst");
  Checkbox poljeZaPotvrdu[] = new Checkbox[4];
  public PrimerPoljaZaPotvrdu() { super("Polja za potvrdu");
    dodajKomponente(); setSize(250,120); setVisible(true);
   }
   void dodajKomponente() {
    add(labela, "North"); labela.setAlignment(Label.CENTER);
    Panel panel = new Panel();
    Panel panel1 = new Panel();
    panel1.setLayout(new GridLayout(2,1));
    Panel panel2 = new Panel();
    panel2.setLayout(new GridLayout(2,1));
    poljeZaPotvrdu[0] = new Checkbox("jedan");
    poljeZaPotvrdu[1] = new Checkbox("dva");
```

Primer polja za potvrdu i radio-dugmadi (2)

```
  CheckboxGroup grupa = new CheckboxGroup();
  poljeZaPotvrdu[2] = new Checkbox("tri",grupa,false);
  poljeZaPotvrdu[3] = new Checkbox("cetiri",grupa,false);
  for(int i=0;i<4;++i)
      poljeZaPotvrdu[i].addItemListener(this);
  for(int i=0;i<2;++i) panel1.add(poljeZaPotvrdu[i]);
  for(int i=2;i<4;++i) panel2.add(poljeZaPotvrdu[i]);
  panel.add(panel1); panel.add(panel2);
  add(panel, "Center");
}
```

Primer polja za potvrdu i radio-dugmadi (3)

```
  public void itemStateChanged(ItemEvent e) {
    String tekst = "";
    for(int i=0;i<4;++i) {
      if(poljeZaPotvrdu[i].getState())
        tekst+=poljeZaPotvrdu[i].getLabel()+" ";
    }
    labela.setText(tekst);
    if(tekst.equals("cetiri ")) dispose();
  }
  public static void main(String args[]){
    PrimerPoljaZaPotvrdu prozor = new PrimerPoljaZaPotvrdu ();
  }
}
```

Liste i padajuće liste

- Klasa `List` realizuje listu
  - iz koje se može izabrati jedan ili više redova
- Metodi klase `List` omogućavaju
  - modifikaciju elemenata liste i
  - upit o njihovom statusu
- Klasa `Choice` realizuje padajuću listu
  - iz koje se može izabrati samo jedan red
- Metodi klase `Choice` omogućavaju
  - modifikaciju elemenata liste i
  - upit o njihovom statusu

Primer liste i padajuće liste (1)

```
import java.awt.*; import java.awt.event.*;
public class PrimerListe extends Frame implements ItemListener{
  Label labela = new Label("Pocetni tekst");
  Choice izbor = new Choice();
  List lista = new List(3,true);
  public PrimerListe() { super("Liste");
    dodajKomponente(); setSize(200,200); setVisible(true);
  }
  void dodajKomponente() {
    add(labela, "North"); labela.setAlignment(Label.CENTER);
    izbor.addItem("jedan"); izbor.addItem("dva"); izbor.addItem("Kraj");
    izbor.addItemListener(this);
    lista.addItem("tri"); lista.addItem("cetiri"); lista.addItem("pet");
    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(panel, "Center");
  }
```

Primer liste i padajuće liste (2)

```
   public void itemStateChanged(ItemEvent e) {
    String tekst = izbor.getSelectedItem() + " ";
    if(tekst.equals("Kraj ")) dispose();
    for(int i=0;i<3;++i)
      if(lista.isIndexSelected(i))
        tekst += lista.getItem(i) + " ";
    labela.setText(tekst);
  }
  public static void main(String args[]){
    PrimerListe prozor = new PrimerListe ();
  }
}
```

Polje za tekst i prostor za tekst

- Klasa `TextComponent` je osnovna klasa za tekst klase
  - `TextField` i
  - `TextArea`
- Klasa `TextField` omogućava prikaz i unos jedne linije teksta
  - može se definisati karakter koji se pojavljuje umesto unošenog teksta (za lozinke)
  - za definisanje alternativnog karaktera koristi se metod `setEchoCharacter()`
- Klasa `TextArea` omogućava prikaz i unos više linija teksta
  - prostor za tekst obezbeđuje horizontalan i vertikalan klizač za proklizavanje teksta
- Metod `setEditable`()
  - omogućava da tekst objekti budu definisani samo za čitanje

Primer polja za tekst i prostora za tekst (1)

```
import java.awt.*;
import java.awt.event.*;
public class PrimerTekst extends Frame implements ActionListener{
  TextField poljeZaTekst = new TextField("Uneti tekst ovde.");
  TextArea prostorZaTekst = new TextArea("I bice umetnut ovde!");
  public PrimerTekst() {
    super("Tekst");
    dodajKomponente();
    setSize(200,150);
    setVisible(true);
  }
  void dodajKomponente() {
   add(poljeZaTekst, "North");
   add(prostorZaTekst, "Center");
   poljeZaTekst.addActionListener(this);
  }
```

Primer polja za tekst i prostora za tekst (2)

```
  public void actionPerformed(ActionEvent e) {
    String tekst = poljeZaTekst.getText();
    if(tekst.equals("Kraj")) dispose();
    prostorZaTekst.insert(tekst+"\n",0);
  }
  public static void main(String args[]){
    PrimerTekst prozor = new PrimerTekst ();
  }
}
```

Platno i crtanje

- Platno (kanvas, `Canvas`) je jednostavna komponenta
  - predstavlja praznu pravougaonu površinu
  - po kojoj može da se crta i na kojoj se hvataju događaji
  - crta se redefinisanjem metode `paint(Graphics)`
- Za crtanje se koriste metode klase `Graphics`
  - primitive: linija, pravougaonik, mnogougao, elipsa, luk, tekst, slika
  - operacije sa prefiksom `draw...`
  - posebne operacije za popunjene primitive, prefiks `fill...`
  - translacija koordinatnog sistema grafičkog konteksta
    - nakon translacije nove primitive se crtaju u novom koordinatnom sistemu
  - postavljanje boje crtanja: `void setColor(Color)`

Primer platna

```
import java.awt.*; import java.awt.event.*;
public class Crtanje extends Frame{
  private class Platno extends Canvas{
    public void paint(Graphics g){int sirina=getWidth(),visina=getHeight();
      g.drawLine(0,0,sirina-1,visina-1);
      g.translate(sirina-1-sirina/3,0);
      g.setColor(Color.RED); g.fillRect(0,0,sirina/3, visina/3);
      g.setColor(Color.GREEN); g.drawRect(0,0,sirina/3, visina/3);
      g.translate(-sirina+1+sirina/3,visina-1-visina/3);
      g.setColor(Color.YELLOW); g.fillOval(0,0, sirina/3, visina/3);
      g.setColor(Color.BLUE); g.drawOval(0,0, sirina/3, visina/3);
    }
  }
  public Crtanje(){
    super("Crtanje"); add(new Platno()); //...
  }
   public static void main(String[] args){ new Crtanje(); }
}
```