# Java GUI

**GUI**

- 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

**Osnovni paketi**

Osnovni elementi GUI se mogu kreirati upotrebom klasa definisanih u paketima-

- **java.awt** – osnovni paket (Abstract Windowing Toolkit) , za većinu komponenti definisanih u ovom paketu je pisan native kod koji ih onda čini platformski zavisnim, tako da GUI aplikacije sastavljene od awt komponenti samo liči na različitim plaformama
- **javax.swing** – većina klasa definisanih u awt paketu je poboljšana/proširena u swing paketu; sve klase pisane isključivo u Javi; Swing klase su deo opštijeg skupa "alata" namenjenih izgradnji GUI-a koji se naziva **JFC-om (Java Foundation Classes).** JFC obuhvata pored Swing komponenti i drag-n-drop capabilty paket, accessibility paket …; Opšta karakteristika Swing klasa je da su fleksibilnije od odgovarajućih awt klasa
- **najava sasvim novog koncepta podrške razvoja GUI – a …**

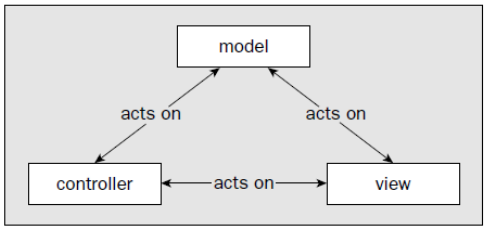
**AWT**

- Abstract Windowing Toolkit - apstraktni alati za rad sa prozorima
  apstraktni: ne zavise od konkretne platforme
- sadrži klase i interfejse koji podržavaju izlazne i ulazne aspekte GUI
- koristi se za programiranje
  - samostalnih aplikacija
  - apleta
- Komponente koje se pojavljuju na ekranu nazivaju se i "kontrole" ili vidgets
  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)

**Model-View-Controller architecture**

Swing komponente su grubo gledano zasnovane na MVC arhitekturi

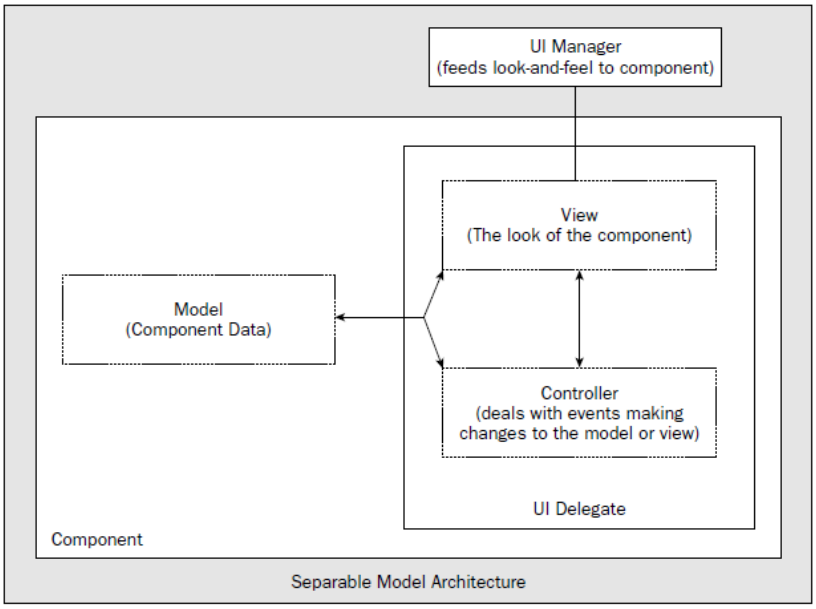
- MVC je prvi put uveden u okviru SmallTalk-a i predstavlja ideju trodelnog modela komponenti:
  - **Model** deo – zadužen za čuvanje podataka kojima je komponenta definisana
  - **View** deo – koji je zadužen za prikaz same komponente
  - **Contoller** deo – zadužen za interakciju sa korisnikom, tačnije kada korisnik ima bilo kakvu "komunikaciju" sa komponentom, Contoller deo registruje akciju i menja Model i/ili View ako je tako nešto na osnovu akcije korisnika potrebno




**Separable Model architecture**

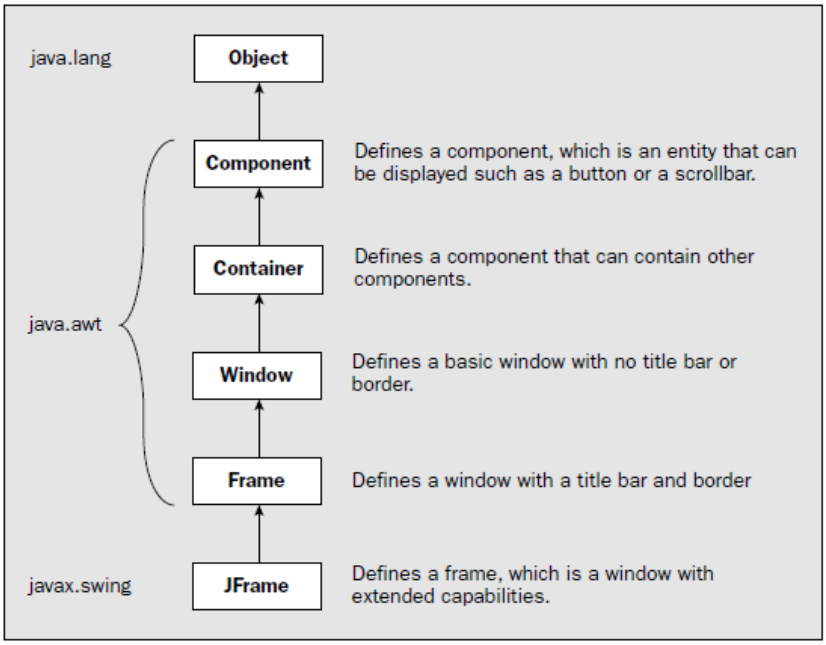
Implementacija MVC arhitekture bi očigledno zahtevala da svaki logički deo bude predstavljen posebnim tipom/klasom. U praksi je ovo teže izvodljivo zbog zavisnosti kontrolera od fizičke reprezenacije komponente (View). Iz tog razloga su View i Controller

predstavljeni/implementirani složenim objektom koji odgovara View-u sa integrisanim kotrolerom.
Zbog navedenog MVC je transformisan u document/view arhitekturu, koju Sun naziva **Separable Model architecture**.




Jedna od "lepih" posledica MVC arhitekture su tzv. **pluggable look-and-feel** komponente, koje bi trebale da imaju osobinu da im se izgled može lako menjati nezavisno od Model i Controller dela.
Swing komponente omogućavaju pluggable look-and-feel korišćenjem nezavisnog objekta nazvanog **UI delegate**, koji predstavlja view+controller deo MVC modela. Različiti UI delegati daju komponenti drugačiji look-and-feel.

## KREIRANJE PROZORA




Primer

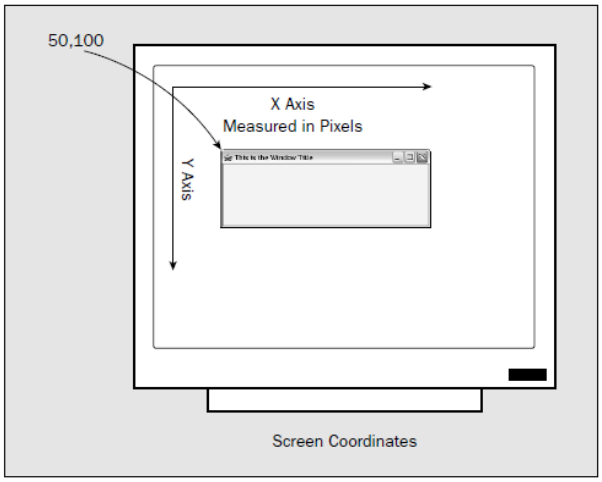
```
import javax.swing.JFrame;
public class TryWindow {
       static JFrame aWindow = new JFrame("This is the Window Title");
       public static void main(String[] args) {
              int windowWidth = 400;           // Window width in pixels
              int windowHeight = 150;    // Window height in pixels
              aWindow.setBounds(50, 100, windowWidth, windowHeight);
```

```
                aWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                aWindow.setVisible(true); // Display the window
            }
        }
```

`aWindow.setBounds(50, 100, windowWidth, windowHeight);`
50, 100 pixel-a od gornjeg levog ugla ekrana
400 x 500 pixel-a veličina prozora
`aWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`
definisanje ponašanja pri zatvaranju prozora (klikom na close dugme prozora ili biranjem close opcije iz kontrolnog menija)

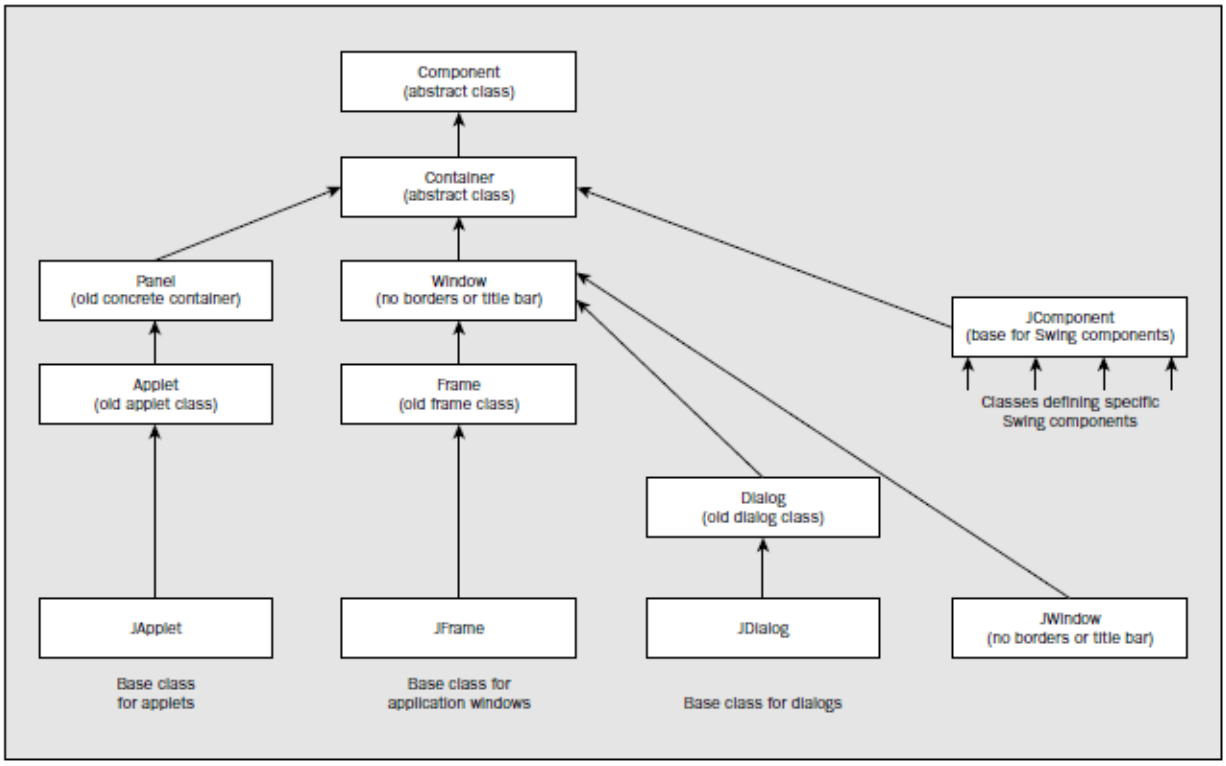
| | |
|---|---|
| EXIT_ON_CLOSE | close the window, dispose of the window resources and those of any components it contains, and finally to terminate the application |
| DISPOSE_ON_CLOSE | This causes the frame and any components it contains to be destroyed but doesn't terminate the application. |
| DO_NOTHING_ON_CLOSE | This makes the close operation for the frame window ineffective. |
| HIDE_ON_CLOSE default | This just hides the window by calling its `setVisible()` method with an argument of false. When a window is hidden, you can always display the window again later by calling `setVisible()` with an argument of `true`. |

`aWindow.setVisible(true);`



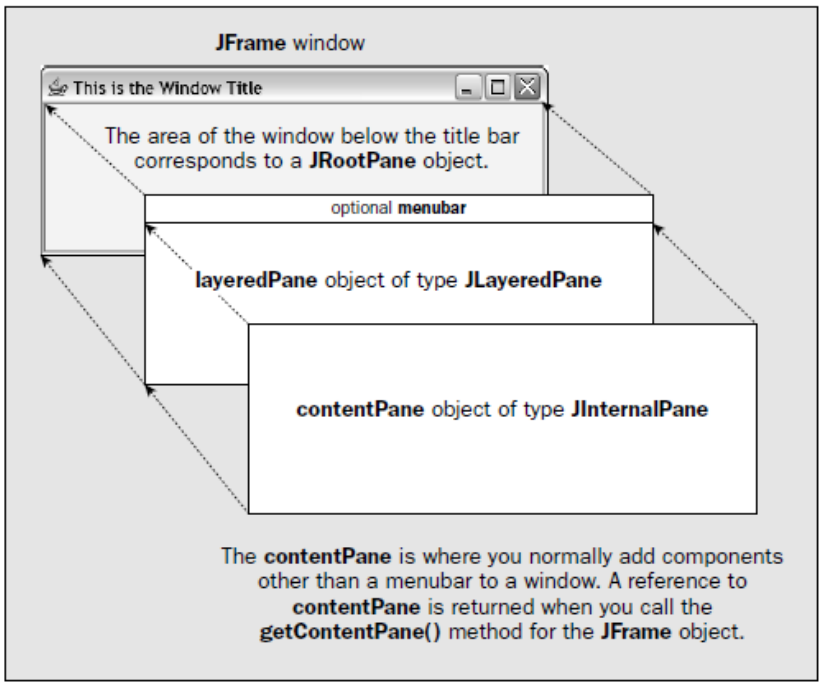
## KOMPONENTE I KONTEJNERI

Komponenta predstavlja grafički element koji je moguće prikazati na ekranu.
Kontejner – komponenta koja u sebi može sadržati druge komponente




The basic difference between a `JFrame` object and a `Window` object is that a `JFrame` object represents the main window for an application, where as a Window object does not — you always need a `JFrame` object before you can create a `Window` object.
Since the `JDialog` class is derived directly from the `Window` class, you can create a `JDialog` object in an application only in the context of a `JFrame` object. Apart from the default constructor, the constructors for the JDialog class generally require a JFrame object to be passed as an argument. This JFrame object is referred to as the parent of the JDialog object.
Since a JFrame object is the top-level window in an application, its size and location are defined relative to the screen. A JDialog object with a JFrame object as a parent will be located relative to its parent.

### Window Panes



When you want to add GUI components or draw in a window displayed from a JFrame object, you add the components to, or draw on, a **window pane** that is managed by the JFrame object. The same goes for an applet. Broadly speaking, window panes are container objects that represent an area of a window, and they come in several different types.
additional pane - glassPane object, which also corresponds to the complete JRootPane area. The contents of the glassPane object displays on top of all the other panes, so this is used to display components that you always want to display on top of anything else displayed in the window—such as drop-down menus. You can also use the glassPane object to display graphics that need to be updated relatively frequently—such as when you create an animation.
A JApplet object has the same arrangement of panes as a JFrame object, so adding components to an applet, or drawing on it, works in exactly the same way. An applet defined as a JApplet object can also have a menu bar just like an application window.

## OSNOVNO o KOMPONENTAMA uopšte

The `Component` class (private) attributes:

- `position` is stored as (x, y) coordinates in the coordinate system of the container object.
- `name` of the component is stored as a String object.
- `size` is recorded as values for the width and the height of the object.
- `foreground color` and `background color`
- `font`
- `cursor` for the object—this defines the appearance of the cursor when it is over the object.
- `enabled` - when a component is enabled, its enabled state is true, and it has a normal appearance. When a component is disabled it is grayed out. Note that a disabled component can still originate events. (To prevent events from a disabled component having an effect, you must call isEnabled() for the component in your event handling code to determine whether the component is enabled or not. You can then choose to do nothing when the isEnabled() method returns false.)
- `visible` on the screen — if an object is not marked as visible, it is not drawn on the screen.
- `Valid` — if an object is not valid, the layout of the entities that make up the object has not been determined. This is the case before an object is made visible. You can make a Container object invalid by changing its contents. It will then need to be validated before it is displayed correctly.

```
myWindow.setName("The Name");
String theName = myWindow.getName();
isVisible(), isEnabled(), and isValid(), setVisible() and setEnabled()
```

### The Size and Position of a Component

| | |
|---|---|
| `Rectangle getBounds()` | Returns the position and size of the object as an object of type Rectangle. |
| `Dimension getSize()` | Returns the current size of the Component object as a Dimension object. |
| `Point getLocation()` | Returns the position of the Component object as an object of type Point. |
| `void setBounds(int x, int y, int width, int height)` | Sets the position of the Component object to thecoordinates (x, y) and the width and height of the object to the values defined by the third and fourth arguments |
| `void setBounds(Rectangle rect)` | Sets the position and size of the Component object to be that of the Rectangle argument rect |
| `void setSize(Dimension d)` | Sets the width and height of the Component object to the values stored in the members of the object d |
| `setLocation(int x, int y)` | Sets the position of the component to the point defined by (x, y) |
| `setLocation(Point p)` | Sets the position of the component to the |

| | |
|---|---|
| | point p |
| | |

* java.awt.Point, java.awt.Dimension, java.awt.Dimension

Primer

```
import javax.swing.JFrame;
import java.awt.Toolkit;
import java.awt.Dimension;
public class TryWindow2 {
	// The window object
	static JFrame aWindow = new JFrame("This is the Window Title");

	public static void main(String[] args) {
		Toolkit theKit = aWindow.getToolkit(); // Get the window toolkit
		Dimension wndSize = theKit.getScreenSize(); // Get screen size
	// Set the position to screen center & size to half screen size
		aWindow.setBounds(wndSize.width/4, wndSize.height/4, // Position
		wndSize.width/2, wndSize.height/2); // Size
		aWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		aWindow.setVisible(true); // Display the window
	}
}
```

Toolkit is an abstract class, getToolKit()for a component gives a reference for Toolkit object.

Primer

```
import javax.swing.JFrame;
import java.awt.Point;
import java.awt.GraphicsEnvironment;
public class TryWindow3 {
	// The window object
static JFrame aWindow = new JFrame("This is the Window Title");

	public static void main(String[] args) {
	Point center = GraphicsEnvironment.getLocalGraphicsEnvironment().
			getCenterPoint();
	int windowWidth = 400;
	int windowHeight = 150;
	// set position and size
	aWindow.setBounds(center.x-windowWidth/2, center.y-windowHeight/2,
		windowWidth, windowHeight);
	aWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	aWindow.setVisible(true); // Display the window
	}
}
```

A `java.awt.GraphicsEnvironment` object contains information about the graphics devices attached to a system, including the display—or displays in systems with more than one.

## Visual Characteristics of a Component

Two things determine the visual appearance of a component: the representation of the component created by the Java code in the component class that is executed when the component is displayed and whatever you draw on the component. You can draw on a Component object by implementing its paint() method. You used this method in Chapter 1 to output the text for our applet. The paint() method is called automatically when the component needs to be drawn.

The following methods have an effect on the appearance of a Component object:

| | |
|---|---|
| `void setBackground(Color aColor)` | Sets the background color to aColor. The background color is the color used for the basic component. |
| `Color getBackground()` | Retrieves the current background color. |
| `void setForeground(Color bColor)` | Sets the foreground color to bColor. The foreground color is the color used for anything appearing on the basic component, such as the label on a button, for example. |

| | |
|---|---|
| `Color getForeground()` | Retrieves the current foreground color |
| `void setCursor(Cursor aCursor)` | Sets the cursor for the component to aCursor. This sets the appearance of the cursor within the area occupied by the Component object. |
| `void setFont(Font aFont)` | Sets the font for the Component object. |
| `Font getFont()` | Returns the Font object used by the component. |

## SWING KOMPONENTE

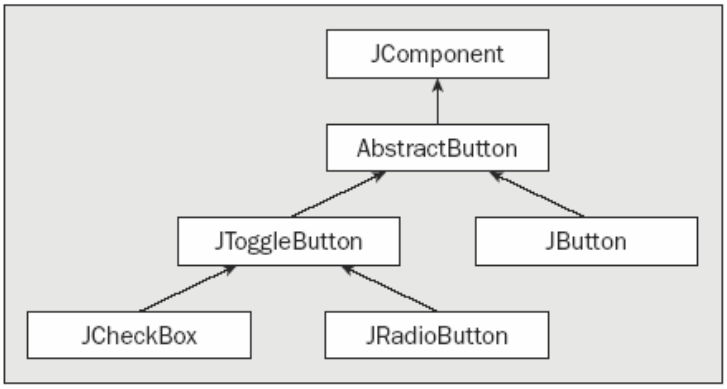
Swing komponente imaju `JComponent` klasu kao osnovnu, a ona nasleđuje klasu `Component` dodajući sledeće mogućnosti:

- Podrška **pluggable look-and-feel za komponente** – mogućnost promene izgleda komponenti koje se prikazuju
- Podrška tooltip-ova, definisani u klasi `JTooltip` – poruka koja opisuje svrhu komponente kada miš stoji nad njom
- Podrška **automatskog skrolovanja** u listi, tabeli, ili drvetu kada se komponenta prevlači mišem

Sve klase Swing komponenti su definisane u paketu `javax.swing` i imaju imena koja počinju sa J.

## Dugmad, Dugmići or Gumbići ☺ (Buttons)

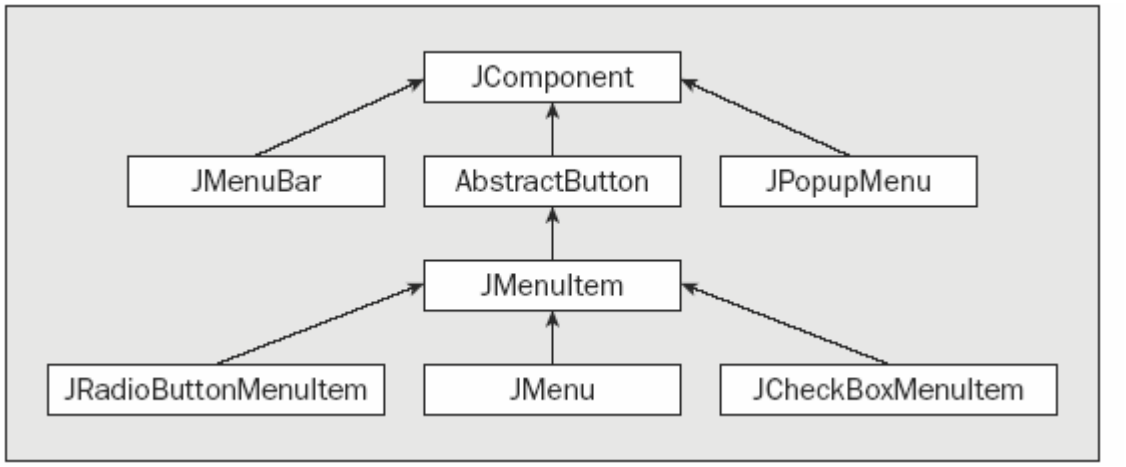
Klase Swing dugmadi definisane su u paketu `javax.swing` i kao osnovnu klasu imaju `AbstractButton` klasu iz koje su ostale izvedene, kao što pokazuje naredna slika:



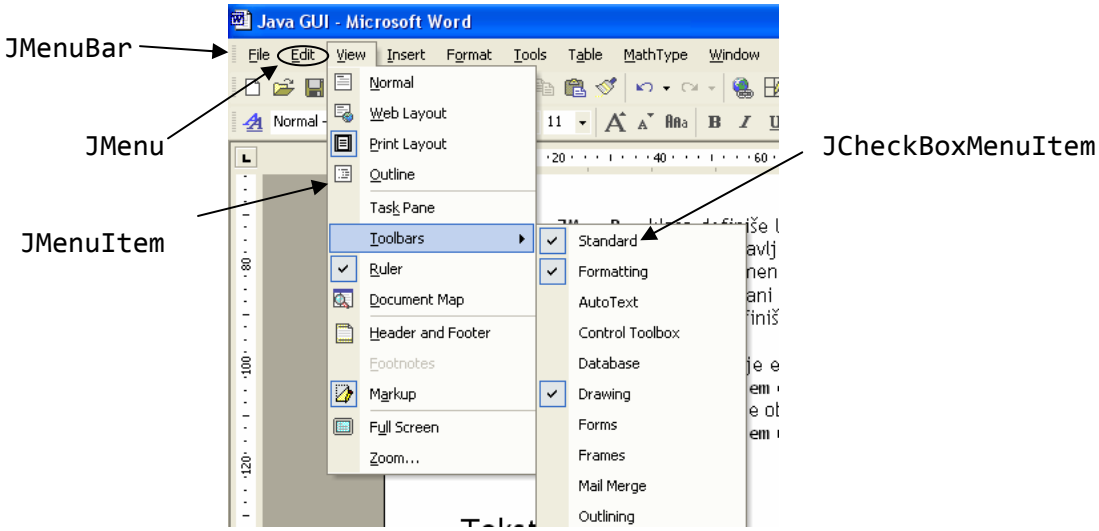

| | | |
|---|---|---|
| `JButton` | regularno dugme, npr. OK/Cancel.U kombinaciji sa klasom JToolBar služi za kreiranje toolbar-a |  |
| `JToggleButton` | definiše dugme sa dva stanja – pritisnuto ili ne, najčešće se koriste njegove specijalizacije (JcheckBox i JRadioButton), a ovu klasu ima smisla koristiti kada je potrebno definisati pojavljivanje koje nije tipičan radio button ili checkbox | |
| `JCheckBox` | dugme sa checkbox-om sa leve strane |  |
| `JRadioButton` | dugmad koja operišu u grupi, tako da je samo jedno dugme pritisnuto u datom trenutku. Ovo grupisanje se postiže dodavanjem `JRadioButton` objekta `ButtonGroup` objektu |  |

# Meniji (Menus)

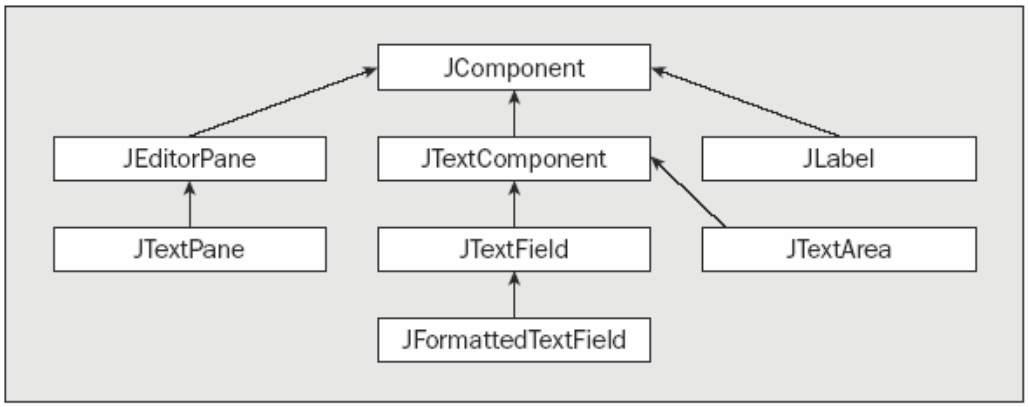
Swing komponente uključuju podršku pop-up menija i kontekstnih menija kao i liniju menija.




- `JMenuBar` klasa definiše liniju menija koja se najčešće nalazi na vrhu prozora aplikacije.
- `JMenu` objekat predstavlja element na vrhu menija. Klikom na njega dobija se padajuća (drop down) lista elemenata menija.
- Stavke menija definisani su klasom `JMenuItem`.
- `JPopupMenu` klasa definiše kontekstni meni koji se najčešće koristi da se prikaže pri desnom kliku na komponentu.
- `JCheckBoxMenuItem` je element menija sa checkbox-om.
- `JRadioButtonMenuItem` definiše element menija koji je deo grupe gde samo jedan element može da bude obeležen u datom trenutku. Grupa se kreira dodavanjem `JRadioButtonMenuItem` objekata ButtonGroup objektu.



# Tekst komponente




| | | |
|---|---|---|
| `JLabel` | pasivna komponenta, ne može se editovati, služi za označavanje drugih komponenti | I am a label |
| `JTextField` | Predstavlja jednu liniju teksta i može se editovati. | 10-Apr-2004 |
| `JFormattedTextField` | JTextField komponenta koja ima kontrolu formata podataka koji se unose/prikazuju | |
| `JTextArea` | komponenta koja dozvoljava unos višelinijskog teksta. Ne podržava direktno skrolovanje, ali to se može postići stavljanjem komponente `JTextArea` u kontejner `JScrollPane` | 'Tis a dog's delight to bark and bite<br>And little birds to sing,<br>And if you sit on a red hot brick,<br>'Tis a sign of an early spring! |
| `JeditorPane` i `JTextPane` | podržavaju sofisticiranije editovanje (HTML, slike itd) | |

# Ostale Swing komponente

| | |
|---|---|
| `JPanel` | definiše vrstu fizičkog panela koji se koristi kao kontejner za grupe ili skup komponenti |
| `JList` | definiše ograničenu listu stavki<br>**Shadrach**<br>**Meshak**<br>**Abednego** |
| `JTable` | definiše tabelu u kojoj se može selektovati kolona, vrsta ili pojedinačni element. Osim toga ova klasa automatski vodi računa o prevlačenju kolona mišem na novu poziciju. |
| `javax.swing.borders` | sadrži osam klasa koje definišu različite vrste granica komponenti |

| First Name | Last Name | Occupation |
|---|---|---|
| Julia | Roberts | Actress |
| Brad | Pitt | Actor |
| Alfred | Hitchcock | Director |

# KONTEJNERI

Kontejner je objekat svakog tipa koji kao osnovnu klasu ima `Container` klasu, pa su sve Swing komponente kontejneri. Komponente unutar kontejnera se prikazuju unutar površi koju zauzima kontejner na ekranu. Takođe, kontejner kontroliše kako su raspoređene njegove komponente pomoću objekta koji nazivamo layout manager. Klasa Container je apstraktna klasa.

**Napomena!** Kontejner ne može da sadrži objekat klase Window, niti bilo koje druge klase izvedene iz nje. Objekat bilo koje klase koja je izedena iz klase Component može biti dodat kontejneru.

Metode klase `Container`:

- `int getComponentCount()` – vraća broj komponenti u kontejneru
- `Component getComponent(int index)` – vraća komponentu koja se identifikuje datim indeksom. Radi se o indeksu niza, pa mora biti u opsegu [0, count-1].
- `Component[] getComponents()` – vraća niz komponenti iz kontejnera

```
Component component = null; // Stores a Component
int numComponents = content.getComponentCount(); // Get the count

for(int i = 0; i < numComponents; i++) {
       component = content.getComponent(i); // Get each component
       // Do something with component...
}
ili
Component[] theComponents = content.getComponents(); // Get all components
for(Component component : theComponents) {
// Do something with component...
}
```

## Dodavanje komponente kontejneru

- `Component add(Component c)` – dodaje komponentu c na kraj liste komponenti sačuvane u kontejneru. Vraća se c.
- `Component add(Component c, int index)` – komponenta se smešta na datu poziciju. Ako je index -1 komponenta se dodaje na kraj liste, u suprotnom indeks mora biti ne manji od 0 i manji od trenutnog broja komponenti u kontejneru.Vraća se c.
- `void add(Component c, Object constraints)` – dodaje komponentu c na kraj liste komponenti sačuvane u kontejneru. Pozicija komponente takođe zavisi i od ograničenja definisanih drugim argumentom (constraints).
- `void add(Component c, Object constraints, int index)` – komponenta se smešta na datu poziciju. Ako je index -1 komponenta se dodaje na kraj liste, u suprotnom indeks mora biti ne manji od 0 i manji od trenutnog broja komponenti u kontejneru. Pozicija komponente takođe zavisi i od ograničenja definisanih drugim argumentom (constaints).

Kada se doda komponenta na datu poziciju, ostale komponente se pomeraju redom da bi nova mogla da se doda. U datom trenutku, komponenta može biti u samo jednom kontejneru. Dodavanje komponente koja je već deo drugog kontejnera ima za posledicu njeno uklanjanje iz tog kontejnera.

# LAYOUT MANAGER

Svi kontejneri imaju podrazumevani layout manager, ali se može izabrati drugačiji kada je potrebno. Mnoge klase koje definišu layout manager su u paketima `java.awt` i `javax.swing`. Layout manager za kontejner određuje poziciju i veličinu svih komponenti u njemu, tako da generalno ne treba menjati veličinu i poziciju komponenti – o tome se stara layout manager. Klase koje definušu layout manager-e imlementiraju intefrejs `LayoutManager`, tako da se promenljiva tipa `LayoutManager` može koristiti kao referenca na bilo koji tip layout manager-a. Neke layout manager klase:

- `FlowLayout` – `java.awt` dodaju se komponente u sukcesivnim redovima – kad je red popunjen, počinje se sa novim. Najčešće se koristi za uređivanje dugmeta. To je podrazumevani LayoutManager za JPanel.

- BorderLayout – java.awt prozor je podeljen na 5 delova – north, south, east, west, center. Ovo je podrazumevani layout za contentPane u JFrame-u, JDialog, JApplet.
- CardLayout – java.awt komponente se raspoređuju jedna iznad druge, kao špil karata. Jedino je komponenta na "vrhu" vidljiva u bilo kom trenutku.
- GridLayout – java.awt komponente se raspoređuju u pravougaonu mrežu, pri čemu korisnik zadaje broj redova i kolona.
- GridBagLayout – java.awt pravougaona mreža, pri čemu širina redova i kolona može da varira.
- BoxLayout – javax.swing raspoređuje komponente u red ili kolonu, pri čemu se iste skraćuju pre nego što pređu u drugi red ili kolonu. BoxLayout manager je podrazumevan za klasu Box kontejnera.
- SpringLayout – javax.swing omogućava pozicioniranje komponente uz ivicu kontejnera ili neke druge komponente

```
FlowLayout flow = new FlowLayout();
aWindow.getContentPane().setLayout(flow);
```

## KREIRANJE MENIJA

- Dodavanje stavke meniju:
  - JMenuItem add(String) - kreira stavku sa datim imenom i dodaje je na kraj datog menija

    ```
    JMenuItem newItem = fileMenu.add("New");
    ```
  - JMenuItem add(JMenuItem)

    ```
    JMenuItem newItem = new JMenuItem("New");
    fileMenu.add(newItem);

    JMenu fileMenu = new JMenu("File");
    JMenuItem openMenu = new JMenuItem("Open");
    JCheckboxMenuItem circleItem = new JCheckboxMenuItem("Circle");
    ```

- Druge metode klase JMenuItem:
  - void setEnabled(boolean b); - ako je b true, stavka je dostupna, inače je nedostupna. Podrazumevano stanje je dostupno.
  - void setText(String label) – postavlja labelu stavke na dati string
  - String getText() – vraća labelu stavke

```
import javax.swing.JFrame;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;

public class SketchFrame extends JFrame {
  private JMenuBar menuBar = new JMenuBar();      // Window menu bar
  public SketchFrame(String title) {
    setTitle(title);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setJMenuBar(menuBar);                         // Add the menu bar to the window
    JMenu fileMenu = new JMenu("File");           // Create File menu
    JMenu elementMenu = new JMenu("Elements");    // Create Elements menu
    menuBar.add(fileMenu);                        // Add the file menu
    menuBar.add(elementMenu);                     // Add the element menu
    JMenuItem openMenu = new JMenuItem("Open");
    fileMenu.add(openMenu);
  }
}
```

```
import java.awt.Toolkit;
import java.awt.Dimension;
public class Sketcher {
  private static SketchFrame window;             // The application window

  public static void main(String[] args) {
    window = new SketchFrame("Sketcher");        // Create the app window
    Toolkit theKit = window.getToolkit();        // Get the window toolkit
    Dimension wndSize = theKit.getScreenSize();  // Get screen size
    window.setBounds(wndSize.width/4, wndSize.height/4,        // Position
                     wndSize.width/2, wndSize.height/2);       // Size
    window.setVisible(true);
  }
}
```

## Postavljanje prečica (shortcut)

Tipična prečica: Alt + slovo iz labele stavke menija.

```
fileMenu.setMnemonic('F');
```

`setMnemonic()` je metod koji se nasleđuje iz klase `AbstractButton`, pa ga sve nasleđene klase imaju.

# DODATAK

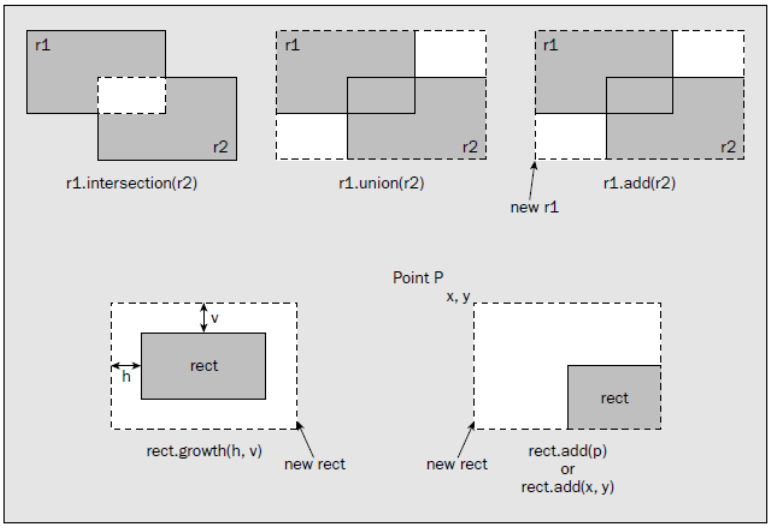
## Point Objects `java.awt.Point`

```
import java.awt.Point;
public class PlayingPoints {
public static void main(String[] args) {
       Point aPoint = new Point(); // Initialize to 0,0
       Point bPoint = new Point(50,25);
       Point cPoint = new Point(bPoint);
       System.out.println("aPoint is located at: " + aPoint);
       aPoint.move(100,50); // Change to position 100,50
       bPoint.x = 110;
       bPoint.y = 70;
       aPoint.translate(10,20); // Move by 10 in x and 20 in y
       System.out.println("aPoint is now at: " + aPoint);
       if(aPoint.equals(bPoint))
       System.out.println("aPoint and bPoint are at the same location.");
}
}
```

## Rectangle Objects `java.awt.Rectangle`

The Rectangle class defines four public data members, all of type int. The position of a Rectangle object is defined by the members x and y, and its size is defined by the members width and height.

| | |
|---|---|
| `Rectangle()` | Creates a rectangle at (0, 0) with zero width and height |
| `Rectangle(int x, int y, int width, int height)` | Creates a rectangle at (x, y) with the specified width and height |
| `Rectangle(int width, int height)` | Creates a rectangle at (0, 0) with the specified width and height |
| `Rectangle(Point p, Dimension d)` | Creates a rectangle at point p with the width and height specified by d |
| `Rectangle(Point p)` | Creates a rectangle at point p with zero width and height |
| `Rectangle(Dimension d)` | Creates a rectangle at (0, 0) with the width and height specified by d |
| `Rectangle(Rectangle r)` | Creates a rectangle with the same position and dimensions as r |




| | |
|---|---|
| `Rectangle intersection(Rectangle r)` | |
| `Rectangle union(Rectangle r)` | |
| `void add(Rectangle r)` | |

| | |
|---|---|
| `void add(Point p)` | Expands the current Rectangle object to enclose the Point object p. The result will be the smallest rectangle that encloses the original rectangle and the point. |
| `void add(int x, int y)` | |
| `void grow(int h, int v)` | Enlarges the current Rectangle object by moving the boundary out from the center by h horizontally and v vertically. |
| `boolean isEmpty()` | Returns true if the width and height members of the current Rectangle object are zero, and false otherwise. |
| `boolean equals(Object rect)` | Returns true if the Rectangle object passed as an argument is equal to the current Rectangle object, and false otherwise. |
| `boolean intersects(Rectangle rect)` | Returns true if the current Rectangle object intersects the Rectangle object passed as an argument, and false otherwise. |
| `boolean contains(Point p)` | |
| `boolean contains(int x, int y)` | |

## **Color** - `java.awt.Color`

Additional package `java.awt.color`

A screen color is represented by an object of class Color. The Color class is used to encapsulate colors in the default sRGB color space or colors in arbitrary color spaces identified by a ColorSpace.
The `Color` class defines a number of standard color constants as `public final static` variables, whose RGB values are

```
WHITE (255, 255, 255)          RED (255, 0, 0)
PINK (255, 175, 175)           LIGHT_GRAY (192, 192, 192)
ORANGE (255, 200, 0)           MAGENTA (255, 0, 255)
GRAY (128, 128, 128)           YELLOW (255, 255, 0)
CYAN (0, 255, 255)             DARK_GRAY (64, 64, 64)
GREEN (0, 255, 0)              BLUE (0, 0, 255)
BLACK (0, 0, 0,)
```

Pr:

```
Color myGreen = new Color(0,200,0);
aWindow.setBackground(Color.PINK);
aWindow.setBackground(myGreen);
```

## **System Colors** `java.awt.SystemColor`

The SystemColor class encapsulates the standard colors that the native operating system uses for displaying various components. The class contains definitions for 24 public final static variables of type SystemColor that specify the standard system colors used by the operating system for a range of GUI components.

```
if(colorA.getRGB() == SystemColor.window.getRGB()) {
// colorA is the window background color...
}
```

Field Summary

static SystemColor desktop
The color rendered for the background of the desktop.
static SystemColor info
The color rendered for the background of tooltips or spot help.
static SystemColor infoText
The color rendered for the text of tooltips or spot help.
static SystemColor menu
The color rendered for the background of menus.
static SystemColor menuText

The color rendered for the text of menus.

static SystemColor scrollbar
The color rendered for the background of scrollbars.

static SystemColor text
The color rendered for the background of text control objects, such as textfields and comboboxes.

static SystemColor textHighlightText
The color rendered for the text of selected items, such as in menus, comboboxes, and text.

static SystemColor textInactiveText
The color rendered for the text of inactive items, such as in menus.

static SystemColor textText
The color rendered for the text of text control objects, such as textfields and comboboxes.

static SystemColor window
The color rendered for the background of interior regions inside windows.

static SystemColor windowBorder
The color rendered for the border around interior regions inside windows.

static SystemColor windowText
The color rendered for text of interior regions inside windows.

## **Cursors** `java.awt.Cursor`

An object of the `java.awt.Cursor` class encapsulates a bitmap representation of the mouse cursor. The Cursor class contains a range of final static constants that specify standard cursor types. You use these to select or create a particular cursor.

The standard cursor types are:

| | | |
|---|---|---|
| DEFAULT_CURSOR | N_RESIZE_CURSOR | NE_RESIZE_CURSOR |
| CROSSHAIR_CURSOR | S_RESIZE_CURSOR | NW_RESIZE_CURSOR |
| WAIT_CURSOR | E_RESIZE_CURSOR | SE_RESIZE_CURSOR |
| TEXT_CURSOR | W_RESIZE_CURSOR | SW_RESIZE_CURSOR |
| HAND_CURSOR | MOVE_CURSOR | |

```
Cursor myCursor = new Cursor(Cursor.TEXT_CURSOR);
```

Pr.

```
import javax.swing.JFrame;
import java.awt.Toolkit;
import java.awt.Dimension;
import java.awt.Color;
import java.awt.Cursor;
public class TryWindow4 {
// The window object
static JFrame aWindow = new JFrame("This is the Window Title");

      public static void main(String[] args){
      Toolkit theKit = aWindow.getToolkit();
      Dimension wndSize = theKit.getScreenSize();
      // Set the position to screen center & size to half screen size
      aWindow.setBounds(wndSize.width/4, wndSize.height/4,
      wndSize.width/2, wndSize.height/2);
      aWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      aWindow.setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
      aWindow.getContentPane().setBackground(Color.PINK);
      aWindow.setVisible(true);
}
}
```

## Font `java.awt.Font, java.awt.FontMetrics`

Klasa Font pravi razliku između karaktera i glifa, tj. grafičke reprezentacije karaktera. Različiti fontovi definišu različite glifove za jedan isti karakter.

```
Font myFont = new Font("Serif", Font.ITALIC, 12);
Font myFont = new Font("Serif", Font.ITALIC + Font.BOLD, 12);
```

Prvi argument je ime fonta, drugi je stil fonta, a treći veličina fonta u pikselima.

- `getStyle()` – vraća stil fonta, tipa int
- `getSize()` – vraća veličinu fonta, tipa int
- `isPlain(), isBold(), isItalic()` – testiranje pojedinačnih stilova, rezultat tipa boolean

It is important to keep in mind that fonts are for presenting characters visually, on the screen or on a printer, for example. Although Java has a built-in capability to represent characters by Unicode codes, it doesn't have any fonts because it doesn't display or print characters itself. The responsibility for this rests entirely with your operating system. Although your Java programs can store strings of Japanese or Tibetan characters, if your operating system doesn't have fonts for these characters you can't display or print them.

Metode klase `GraphicsEnvironment` korisne pri radu sa fontovima:

- `getAllFonts()` – vraća niz Font objekata koji egzistiraju na sistemu
- `getAvailableFontFamilyNames()` – vraća niz objekata tipa String koji predstavljaju imena dostupnih fontova

```
GraphicsEnvironment e = GraphicsEnvironment.getLocalGraphicsEnvironment();
Font[] fonts = e.getAllFonts(); // Get the fonts
String[] fontnames = e.getAvailableFontFamilyNames();
```

Sledeće metode omogućavaju promenu već postojećih fontova:

- `deriveFont(int Style)` – kreira novi Font objekat određenog stila (PLAIN, BOLD, ITALIC, ili BOLD+ITALIC)
- `deriveFont(float size)` - kreira novi Font objekat određene veličine
- `deriveFont(int Style, float size)` – kreira novi Font objekat određenog stila i veličine

```
Font newFont = fonts[fonts.length-1].deriveFont(12.0f);
```

Primer.

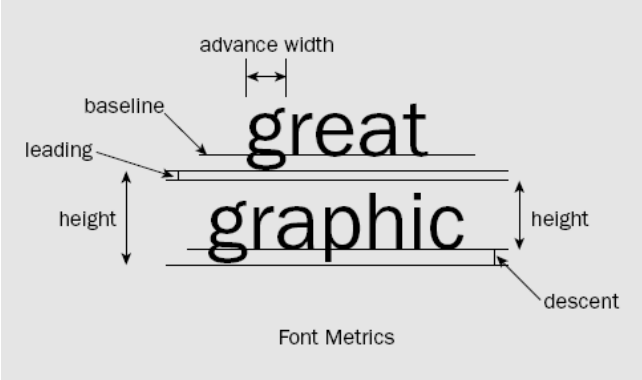
```
import java.awt.Toolkit;
import java.awt.GraphicsEnvironment;
import java.awt.Font;
import java.awt.Dimension;
public class FontInfo {

    public static void main(String[] args) {
       Toolkit theKit = Toolkit.getDefaultToolkit();
       System.out.println("\nResolution: "+ theKit.getScreenResolution() + " dots per inch");
       Dimension screenDim = theKit.getScreenSize();
       System.out.println("Scr Size: "+screenDim.width+" by "+screenDim.height + " pixels");
       GraphicsEnvironment e = GraphicsEnvironment.getLocalGraphicsEnvironment();
       String[] fontnames = e.getAvailableFontFamilyNames();
       System.out.println("\nFonts available on this platform: ");
       int count = 0;
       for (String fontname : fontnames) {
                    System.out.printf("%-30s", fontname);
                    if(++count % 3 == 0) { System.out.println();}
                    }
       return;
       }
}
```

**Metrika fonta**

Svaka komponenta ima metod `getFontMetrics()` koji vraća metriku fonta kao skup podataka o dimenzijama fonta. Argument ovog metoda je objekat tipa Font, a rezultat je tipa FontMetrics.

```
FontMetrics metrics = aWindow.getFontMetrics(aWindow.getFont());
```

Metodi FontMetrics objekata: `getAscent()`, `getMaxAscent()`, `getDescent()`, `getMaxDescent()`, `getLeading()`, `getHeight()`. Svi vraćaju rezultat tipa `int` koji predstavlja odgovarajuću dužinu




Advance width nekog karaktera se dobija pozivom metoda charWidth(), čiji je argument karakter, npr:

```
int widthX = metrics.charWidth('X');
```

Metod getWidths() vraća niz vrednosti tipa int koje predstavljaju širinu svih znakova određenog fonta:

```
int[] widths = metrics.getWidths();
```

Željenom članu ovakvog niza može se pristupiti preko indeksa, ali i navođenjem karaktera čija se širina želi umesto indeksa niza, npr. `widths['X']`.
Postoji i metod `stringWidth()`, čiji je argument objekat tipa String, a povratna vrednost, ukupna širina stringa, tipa int.