MODULE 5

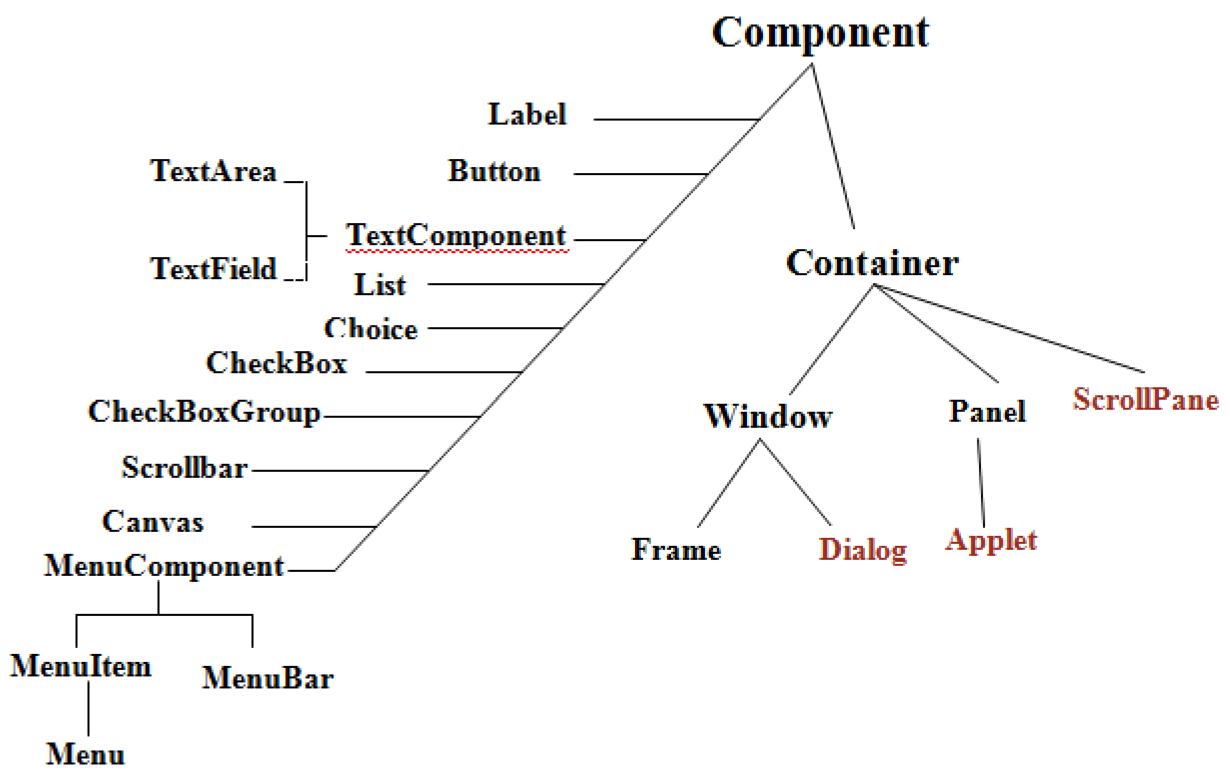
5.1 GRAPHICS PROGRAMMING

5.1.1 Introduction

- **Java AWT** (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.
- The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

5.1.2 Java AWT Hierarchy

The hierarchy of Java AWT classes is given below.



Component

Component is an abstract class that encapsulates all of the attributes of a visual component. It is a super class of all user interface classes. A component is something that can be displayed on a two-dimensional screen and with which the user can interact. Attributes of a component include a size, a location, foreground and background colors, whether or not visible etc.

Container

Container class is a subclass of Component class. This is a type of component that can nest other components within it. Ex:- Window, Frame, and panel are examples of containers. A Container is responsible for laying out (that is, positioning) any components that it contains.

Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Frame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Panel

The **Panel** class is a concrete subclass of **Container** class. A panel is a window that does not contain a title bar, or border. It provides space in which an application can attach any other component, including other panels. It is the superclass for Applet. Other components can be added to a Panel object by its **add()** method. The default layout manager for a panel is the FlowLayout layout manager.

5.1.3 Frames

Frame is a window that is not contained inside another window. Frame is the basis to contain other user interface components in Java graphical applications.

The Frame class can be used to create windows. Frame's constructors:

Frame( )

Frame(String *title*)

After a frame window has been created, it will not be visible until you call setVisible(true).

Setting the Window's Dimensions

The setSize( ) method is used to set the dimensions of the window.

void setSize(int newWidth, int newHeight)

void setSize(Dimension newSize)

The new size of the window is specified by *newWidth* and *newHeight*, or by the width and height fields of the Dimension object passed in *newSize*. The dimensions are specified in terms of pixels.

The getSize( ) method is used to obtain the current size of a window.

Dimension getSize( )

This method returns the current size of the window contained within the width and height fields of a Dimension object.

Hiding and Showing a Window

After a frame window has been created, it will not be visible until you call setVisible( ).

void setVisible(boolean visibleFlag)

The component is visible if the argument to this method is true. Otherwise, it is hidden.

Setting a Window's Title

You can change the title in a frame window using setTitle( ), which has this general form:

void setTitle(String newTitle)

Here, *newTitle* is the new title for the window.

Closing a Frame Window

When using a frame window, your program must remove that window from the screen when it is closed, by calling setVisible(false). To intercept a window-close event, you must implement the windowClosing( ) method of the WindowListener interface. Inside windowClosing( ), you must remove the window from the screen. The example in the next section illustrates this technique.

Examples on frames:

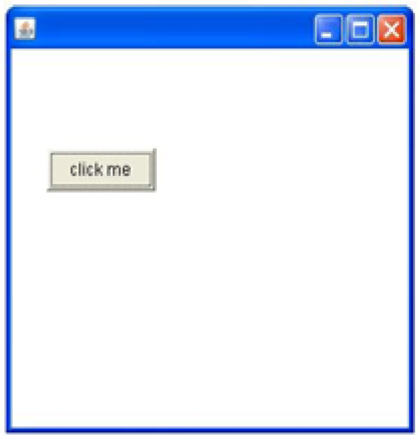
To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

- By extending Frame class (inheritance)
- By creating the object of Frame class (association)

Program for Button component on the Frame(1st method)

```
import java.awt.*;
class First extends Frame{
First(){
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position
add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
setLayout(null);//no layout manager
setVisible(true);//now frame will be visible, by default not visible
}
public static void main(String args[]){
First f=new First();
}}
```

OUTPUT:



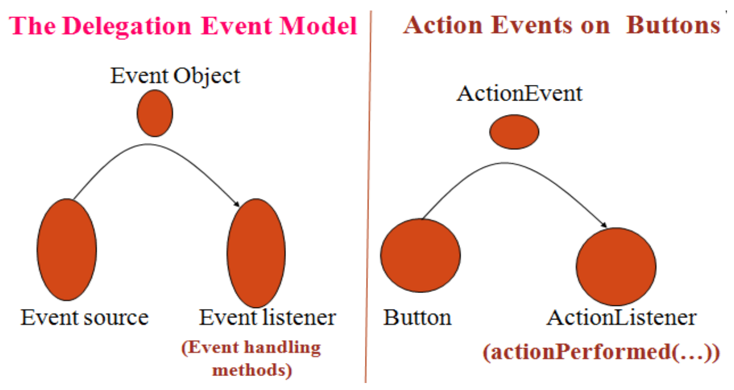
Program on Button component on the Frame(2nd method)

```
import java.awt.*;
class First2{
First2(){
Frame f=new Frame();
Button b=new Button("click me");
b.setBounds(30,50,80,30);
f.add(b);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[]){
First2 f=new First2();
}}
```

5.1.4 Event Driven Programming

5.1.4.1 Delegation Event Model

- It defines standard and consistent mechanisms to generate and process events.
- In this model, a source generates an event and sends it to one or more listeners.
- The listener simply waits until it receives an event. Once received, the listener processes the event and then returns.
- Listeners are created by implementing one or more of the interfaces defined by the java.awt.event package.
- When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument.
- In the delegation event model, listeners must register with a source in order to receive an event notification



Example :

```
class MyActionListener implements ActionListener {
                //Registration Method
                source.addActionListener(this);
                public void actionPerformed(ActionEvent ae){
                        // Handler_ code
                }
}
```

Events

Events are supported by the java.awt.event package.

An event is an object that describes a state change in a source.

Events can be generated as a consequence of a person interacting with elements in a graphical user interface.

Events may also occur that are not directly caused by interactions with a user interface.

For example, an event may be generated

- ✓ when a timer expires,
- ✓ a counter exceeds a value,
- ✓ a software or hardware failure occurs.

Event Sources

A **source** is an object that generates an event. This occurs when the internal state of that object changes in some way.

Sources may generate more than one type of event.

A source must register listeners in order for the listeners to receive notifications about a specific type of event.

Each type of event has its own registration method.

Here is the general form:

public void add*Type*Listener(*Type*Listener *el*)

Here, ***Type*** is the name of the **event** and ***el*** is a reference to the **event listener**.

For example, the method that registers a keyboard event listener is called **addKeyListener( )**.

Event Listeners

A listener is an object that is notified when an event occurs.It has two major requirements.

First, it must have been registered with one or more sources to receive notifications about specific types of events.

Second, it must implement methods to receive and process these notifications.

For example, the MouseMotionListener interface defines two methods to receive notifications when the mouse is dragged or moved.

Any object may receive and process one or both of these events if it provides an implementation of this interface.

5.1.4.2 Event Classes

EventObject is a superclass of all events which is defined in **java.util** package.
At the root of the Java event class hierarchy is EventObject, which is in java.util. It is the superclass for all events. Its one constructor is shown here:
EventObject(Object src)
Here, *src* is the object that generates this event.

AWTEvent is a superclass of all AWT events that are handled by the delegation event model which is defined in **java.awt** package.

EventObject defines two methods: **getSource( )** and **toString( )**. The **getSource( )** method returns the source of the event. Its general form is shown here:
Object getSource( )

The following are the main event classes in **java.awt.event** package.

Event Class	**Description**
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
KeyEvent	Generated when the key is pressed , key is released, or key is typed
TextEvent	Generated when the value of a text area or text field is changed.
MouseWheelEvent	Generated when the mouse wheel is moved
WindowEvent	Generated when a window is activated, deactivated, deiconified, iconified, opened, closing, closed.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
FocusEvent	Generated when a component gains or loses keyboard focus.
AdjustmentEvent	Generated when a scroll bar is manipulated
ContainerEvent	Generated when a component is added to or removed from a container.

.
The ActionEvent Class
You can obtain the command name for the invoking **ActionEvent** object by using the
String getActionCommand() method.
The **getWhen( )** returns the time at which the event took place.

The KeyEvent Class
int getKeyChar( ) which returns the character that was entered, and **int getKeyCode( )** which returns the key code.
It defines many integer constants:

VK_ENTER VK_ESCAPE VK_CANCEL VK_UP
VK_DOWN VK_LEFT VK_RIGHT VK_PAGE_DOWN
VK_PAGE_UP VK_SHIFT VK_ALT VK_CONTROL

The MouseEvent Class
There are eight types of mouse events.
MOUSE_CLICKED MOUSE_DRAGGED MOUSE_ENTERED MOUSE_EXITED
MOUSE_MOVED MOUSE_PRESSED MOUSE_RELEASED MOUSE_WHEEL
int getX( ) int getY( )
The **int getClickCount( )** method obtains the number of mouse clicks for this event.

The ItemEvent Class
The **int getStateChange()** method returns the state change (i.e., **SELECTED** or **DESELECTED**) for the event.
The **Object getItem( )** method can be used to obtain a reference to the item that generated an event.

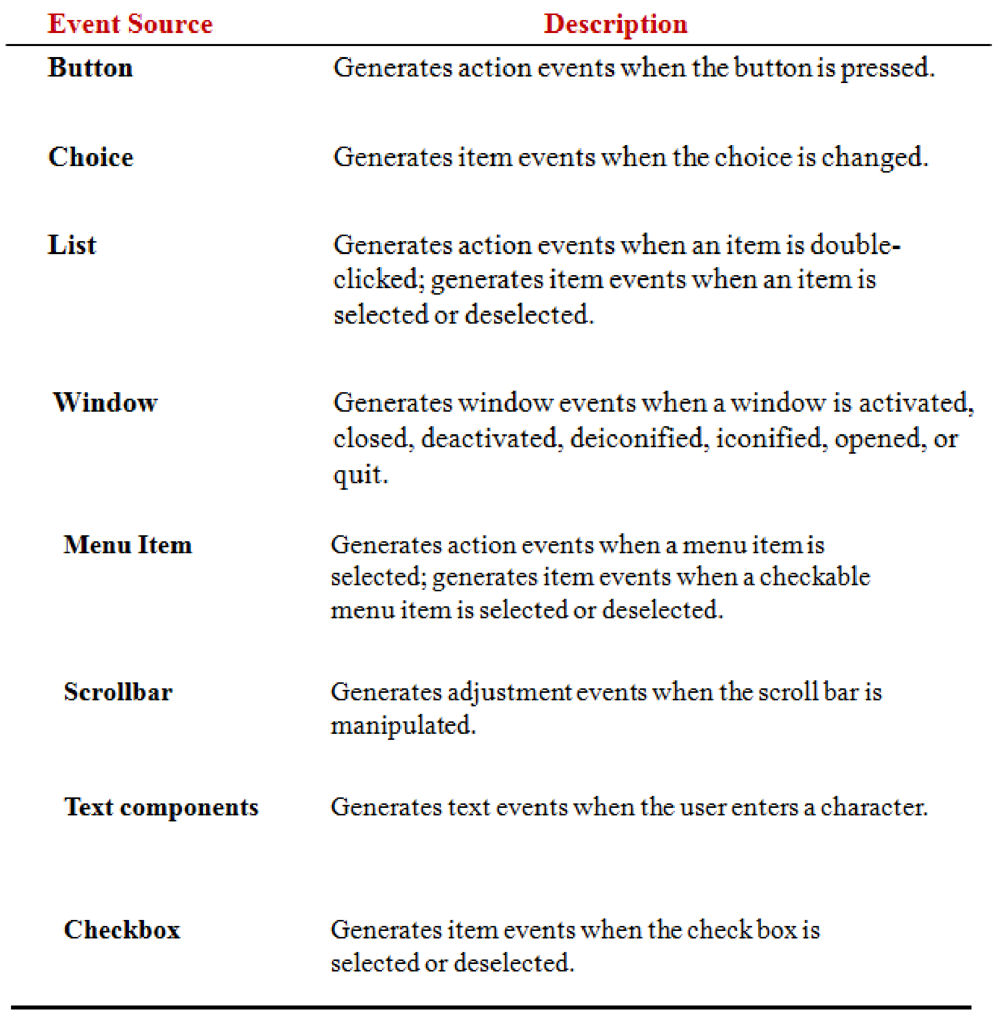
The WindowEvent Class
There are ten types of window events.

The **WindowEvent** class defines integer constants that can be used to identify them. The constants and their meanings are shown here:
WINDOW_ACTIVATED WINDOW_CLOSED WINDOW_CLOSING
WINDOW_DEACTIVATED WINDOW_DEICONIFIED WINDOW_GAINED_FOCUS
WINDOW_ICONIFIED WINDOW_LOST_FOCUS WINDOW_OPENED
WINDOW_STATE_CHANGED

WindowEvent is a subclass of **ComponentEvent**. It defines several constructors. The first is
WindowEvent(Window src, int type)
Here, *src* is a reference to the object that generated this event. The type of the event is *type*.

Source of Events

The following table lists some of the user interface components that can generate the events.



Event Source	Description
Button	Generates action events when the button is pressed.
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.
Menu Item	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Scrollbar	Generates adjustment events when the scroll bar is manipulated.
Text components	Generates text events when the user enters a character.
Checkbox	Generates item events when the check box is selected or deselected.

5.1.4.3 Event Listener Interfaces

Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.

Interface	Description
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved.
TextListener	Defines one method to recognize when a text value changes.
WindowFocusListener	Defines two methods to recognize when a window gains or loses input focus.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Listener Interfaces and associated Event Classes

ActionEvent	**ActionListener**
MouseEvent	**MouseListener** **MouseMotionListener**
KeyEvent	**KeyListener**
TextEvent	**TextListener**
AdjustmentEvent	**AdjustmentListener**
ContainerEvent	**ContainerListener**
FocusEvent	**FocusListener**
ItemEvent	**ItemListener**
TextEvent	**TextListener**
WindowEvent	**WindowListener**

The ActionListener Interface

This interface defines the **actionPerformed()** method that is invoked when an action event occurs. Its general form is shown here:

void actionPerformed(ActionEvent *ae*)

The AdjustmentListener Interface

void adjustmentValueChanged(AdjustmentEvent *ae*)

The ComponentListener Interface

This interface defines four methods that are invoked when a component is resized, moved, shown, or hidden.

void componentResized(ComponentEvent *ce*)
void componentMoved(ComponentEvent *ce*)
void componentShown(ComponentEvent *ce*)
void componentHidden(ComponentEvent *ce*)

The ItemListener Interface

void itemStateChanged(ItemEvent *ie*)

The ContainerListener Interface

When a component is added or removed to and from a container the following methods are invoked.

void componentAdded(ContainerEvent *ce*)
void componentRemoved(ContainerEvent *ce*)

The KeyListener Interface

void keyPressed(KeyEvent *ke*)
void keyReleased(KeyEvent *ke*)
void keyTyped(KeyEvent *ke*)

The MouseListener Interface

void mouseClicked(MouseEvent *me*)
void mouseEntered(MouseEvent *me*)
void mouseExited(MouseEvent *me*)
void mousePressed(MouseEvent *me*)
void mouseReleased(MouseEvent *me*)

The MouseMotionListener Interface

void mouseDragged(MouseEvent *me*)
void mouseMoved(MouseEvent *me*)

The WindowListener Interface

void windowActivated(WindowEvent *we*)
void windowClosed(WindowEvent *we*)
void windowClosing(WindowEvent *we*)
void windowDeactivated(WindowEvent *we*)
void windowDeiconified(WindowEvent *we*)
void windowIconified(WindowEvent *we*)
void windowOpened(WindowEvent *we*)

The MouseWheelListener Interface

void mouseWheelMoved(MouseWheelEvent *mwe*)

The TextListener Interface

void textChanged(TextEvent *te*)

5.1.5 Layout Managers

Layout means the arrangement of components within the container. In other way we can say that placing the components at a particular position within the container. The task of layouting the controls is done automatically by the Layout Manager.

The **layout manager** automatically positions all the components within the container. If we do not use layout manager then also the components are positioned by the default layout manager. It is possible to layout the controls by hand but it becomes very difficult because of the following two reasons.

- It is very tedious to handle a large number of controls within the container.
- Oftenly the width and height information of a component is not given when we need to arrange them.

AWT Layout Manager Classes:

Following is the list of commonly used controls while designed GUI using AWT.

S. No.	LayoutManager & Description
1	**BorderLayout** The borderlayout arranges the components to fit in the five regions: east, west, north, south and center.
2	**CardLayout** The CardLayout object treats each component in the container as a card. Only one card is visible at a time.
3	**FlowLayout** The FlowLayout is the default layout.It layouts the components in a directional flow.
4	**GridLayout** The GridLayout manages the components in form of a rectangular grid.
5	**GridBagLayout** This is the most flexible layout manager class.The object of GridBagLayout aligns the component vertically,horizontally or along their baseline without requiring the components of same size.

5.1.5.1 BorderLayout:

The Border Layout manager arranges components into five regions: **North, South, East, West,** and **Center**. Components in the North and South are set to their natural heights and horizontally stretched to fill the entire width of the container. Components in the East and West are set to their natural widths and stretched vertically to fill the entire width of the container. The Center component fills the space left in the center of the container.If one or more of the components, except the Center component, are missing then the rest of the existing components are stretched to fill the remaining space in the container.

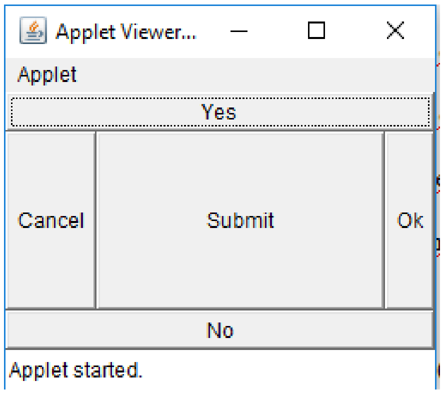
1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

Constructors of BorderLayout class:

BorderLayout()	//No vertical or horizontal gaps.
BorderLayout(hgap, vgap)	//hgap – horizontal gaps between components
	//vgap – vertical gaps between components

Example of BorderLayout class:

```
// Demonstrate Buttons

import java.awt.*;

import java.awt.event.*;

import java.applet.*;

/* <applet code="ExBL" width=250 height=150> </applet>*/

public class ExBL extends Applet{

        String msg="hello";

        Button yes, no,ok,cancel,submit;
public void init() {
        setLayout(new BorderLayout());
        yes = new Button("Yes");
        no = new Button("No");
        ok = new Button("Ok");
        cancel=new Button("Cancel");
        submit=new Button("Submit");
        add(yes,BorderLayout.NORTH);
        add(no,BorderLayout.SOUTH);
        add(ok,BorderLayout.EAST);
        add(cancel,BorderLayout.WEST);
        add(submit,BorderLayout.CENTER);
}
public void paint(Graphics g) {
        g.drawString(msg, 6, 100);
}}
```



Applet started.

5.1.5.2 Grid Layout

The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

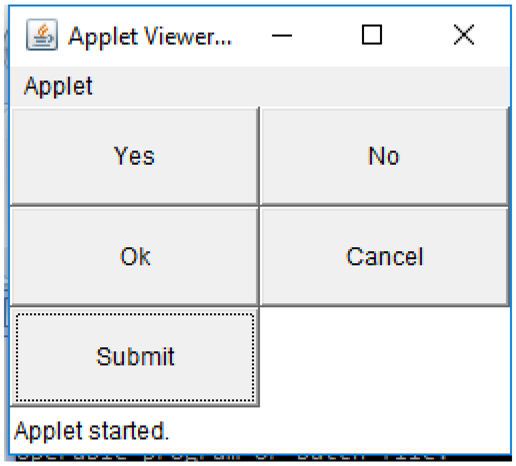
Constructors of GridLayout class:

- GridLayout(): creates a grid layout with one column per component in a row.
- GridLayout(int rows, int columns): creates a grid layout with the given rows and columns but no gaps between the components.
- GridLayout(int rows, int columns, int hgap, int vgap): creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

Example of GridLayout class:

```
// Demonstrate Buttons

import java.awt.*;

import java.awt.event.*;

import java.applet.*;

/* <applet code="ExGL" width=250 height=150> </applet>*/

public class ExGL extends Applet{

    String msg="hello";

    Button yes, no,ok,cancel,submit;

    public void init() {

            //setLayout(new GridLayout());

            setLayout(new GridLayout(3,2));

            yes = new Button("Yes");

            no = new Button("No");

            ok = new Button("Ok");

            cancel=new Button("Cancel");

            submit=new Button("Submit");
```

```
        add(yes);

        add(no);

        add(ok);

        add(cancel);

        add(submit);

    }

    public void paint(Graphics g) {

        g.drawString(msg, 6, 100);

    }
```



5.1.5.3 FlowLayout

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel. When the container is not wide enough to display all the components, the remaining components are placed in the next row, etc. By default each row is centered.

Fields of FlowLayout class:

- public static final int LEFT
- public static final int RIGHT
- public static final int CENTER

- public static final int LEADING
- public static final int TRAILING

Constructors of FlowLayout class:

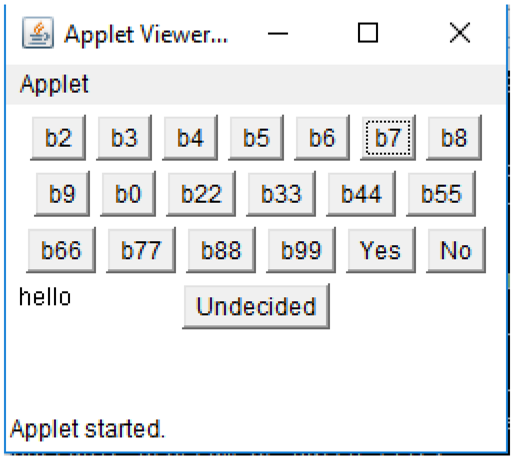
- FlowLayout(): creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
- FlowLayout(int align): creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
- FlowLayout(int align, int hgap, int vgap): creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example of FlowLayout class:

```
// Demonstrate Buttons

import java.awt.*;

import java.awt.event.*;

import java.applet.*;

/* <applet code="ExFL" width=250 height=150> </applet>*/

public class ExFL extends Applet{

    String msg="hello";

    Button yes, no, maybe,b2,b3,b4,b5,b6,b7,b8,b9,b0,b22,b33,b44,b55,b66,b77,b88,b99;

    public void init() {

            yes = new Button("Yes");

            no = new Button("No");

            maybe = new Button("Undecided");

                    b2=new Button("b2");

                    b3=new Button("b3");

                    b4=new Button("b4");

                    b5=new Button("b5");

                    b6=new Button("b6");
```

```
b7=new Button("b7");
b8=new Button("b8");
b9=new Button("b9");
b0=new Button("b0");
b22=new Button("b22");
b33=new Button("b33");
b44=new Button("b44");
b55=new Button("b55");
b66=new Button("b66");
b77=new Button("b77");
b88=new Button("b88");
b99=new Button("b99");
add(b2);
add(b3);
add(b4);
add(b5);
add(b6);
add(b7);
add(b8);
add(b9);
add(b0);
add(b22);
add(b33);
add(b44);
add(b55);
```

```
                              add(b66);

                              add(b77);

                              add(b88);

                              add(b99);

                              add(yes);

                              add(no);

                              add(maybe);

    }

        public void paint(Graphics g) {

                g.drawString(msg, 6, 100);

        }

    }
```



5.1.5.4 CardLayout

CardLayout places components (usually panels) on top of each other in a stack like a deck of cards. You can see only one card at a time. By default, the first card is visible. We can put a cards on top using another control by using the methods next(), previous(), first(), last(), and show().

Constructors of CardLayout class:

- CardLayout(): creates a card layout with zero horizontal and vertical gap.

- CardLayout(int hgap, int vgap): creates a card layout with the given horizontal and vertical gap.

Commonly used methods of CardLayout class:

- public void next(Container parent): is used to flip to the next card of the given container.
- public void previous(Container parent): is used to flip to the previous card of the given container.
- public void first(Container parent): is used to flip to the first card of the given container.
- public void last(Container parent): is used to flip to the last card of the given container.
- public void show(Container parent, String name): is used to flip to the specified card with the given name.

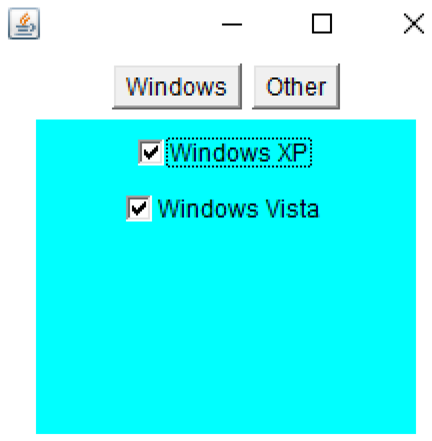
Example of CardLayout class:

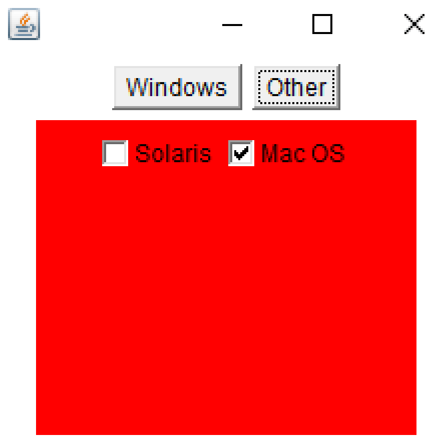
```
// CardLayout demo

import java.awt.*;

import java.awt.event.*;

public class CardLayoutDemo extends Frame implements ActionListener, MouseListener {

    Checkbox winXP, winVista, solaris, mac;

    Button Win, Other;

    Panel osCards;

    CardLayout cardLO;

    public CardLayoutDemo(){

            setLayout(new BorderLayout());

            cardLO = new CardLayout();

            Panel tabs=new Panel();

            Win = new Button("Windows");

            Other = new Button("Other");
```

```
tabs.add(Win);

tabs.add(Other);

add(tabs,BorderLayout.NORTH);

osCards = new Panel();

osCards.setLayout(cardLO); // set panel layout to card layout

Panel winPan = new Panel();

winXP = new Checkbox("Windows XP", null, true);

winVista = new Checkbox("Windows Vista");

// add Windows check boxes to a panel

winPan.setBackground(Color.CYAN);

winPan.add(winXP);

winPan.add(winVista);

// Add other OS check boxes to a panel

Panel otherPan = new Panel();

solaris = new Checkbox("Solaris");

mac = new Checkbox("Mac OS");

otherPan.setBackground(Color.RED);

otherPan.add(solaris);

otherPan.add(mac);

// add panels to card deck panel

osCards.add(winPan, "Windows");

osCards.add(otherPan, "Other");

// add cards to main applet panel

add(osCards,BorderLayout.CENTER);

// register to receive action events
```

```
Win.addActionListener(this);

Other.addActionListener(this);


// register mouse events

addMouseListener(this);

setSize(250,250);

setVisible(true);

}

public Insets getInsets(){

        return new Insets(30,30,30,30);

}

// Cycle through panels.

public void mousePressed(MouseEvent me) {

                cardLO.next(osCards);

}

// Provide empty implementations for the other MouseListener methods.

public void mouseClicked(MouseEvent me) {

}

public void mouseEntered(MouseEvent me) {

}

public void mouseExited(MouseEvent me) {

}

public void mouseReleased(MouseEvent me) {

}

public void actionPerformed(ActionEvent ae) {
```

```
                if(ae.getSource() == Win) {
                        cardLO.first(osCards);
                        //cardLO.show(osCards, "Windows");
                }
                else {
                                cardLO.last(osCards);
                        //cardLO.show(osCards, "Other");
                }
        }
        public static void main(String arg[]){
                new CardLayoutDemo();
        }
}
```





5.1.5.5 GridBagLayout

Flexible GridBagLayout

- Components can vary in size
- Components can occupy multiple rows and columns
- Components can be added in any order

There are two classes that are used:

GridBagLayout: Provides the overall layout manager.

GridBagConstraints: Defines the properties of each component in the grid such as placement, dimension and alignment

Specifying a GridBagLayout

To use a gridbag layout, you must declare the GridBagLayout and GridBagConstaraints object and attach the layout to your applet.

Example:

```
GridBagLayout gridbag = new GridBagLayout();

GridBagConstraints constraints = new GridBagConstraints();

setLayout(gridbag);
```

Setting Constraints for Components

For each component that you add to the gridbag layout, you must first set an instance of the GridBagContraints class.

Let us look at an **example:**

```
constraints.gridx = 0    // put component in first cell.

constraints.gridy = 0

gridbag.setConstraints(button1, constraints);          //set the constraints to button1.

add(button1);                        //add to the Frame
```

GridBagConstraints Data Members

The following are the constraints to control the component:

gridx, gridy, gridwidth, gridheight, weightx, weighty, ipadx, ipady , fill

constraints.gridx=0; //To set the column (Default is RELATIVE).

constraints.gridy=0; //To set the row (Default is RELATIVE).

constraints.gridwidth = 2; //Joins two cells horizontally

constraints.gridheight = 2; //Joins two cells vertically

//Default value is 1.

constraints.weightx=2; //Determines horizontal spacing between cells.

constraints.weighty=3; // Determines vertical spacing between cells.

//Default value is 0.

constraints.ipadx=2; //Specifies extra horizontal space that surrounds a component within cell.

constraints.ipady=0; //Specifies extra vertical space that surrounds a component within a cell.

//Default is 0.

fill //Resizing rules for a component smaller than its display area;

//HORIZONTAL makes component as wide as the display area;

//VERTICAL as tall; BOTH expands it vertically and horizontally.

//NONE (default) is default.

Examples:

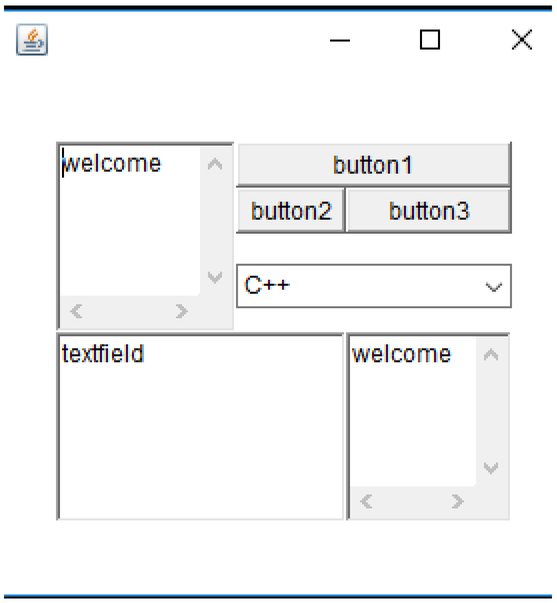
```
constraints.fill = GridBagConstraints.BOTH;          //Fill cell height & width
constraints.fill = GridBagConstraints.HORIZONTAL;        //Fill cell width
```

Example Program

```
import java.awt.*;
class GridBagDemo extends Frame{
        GridBagDemo(){
                GridBagLayout layout=new GridBagLayout();
                   setLayout(layout);
                   GridBagConstraints constraints=new GridBagConstraints();
                   TextArea ta=new TextArea("welcome",5,10);
                   constraints.gridx=0;
                   constraints.gridy=0;
                   constraints.gridwidth=1;
                   constraints.gridheight=3;

                   layout.setConstraints(ta,constraints);

                   add(ta);

                   Button b1=new Button("button1");

                   constraints.fill=GridBagConstraints.BOTH;

                   constraints.gridx=1;

                   constraints.gridy=0;

                   constraints.gridwidth=2;

                   constraints.gridheight=1;

                   layout.setConstraints(b1,constraints);

                   add(b1);

                   Button b2=new Button("button2");

                   constraints.gridx=1;

                   constraints.gridy=1;

                   constraints.gridwidth=1;

                   constraints.gridheight=1;

                   layout.setConstraints(b2,constraints);

                   add(b2);

                   Button b3=new Button("button3");

                   constraints.gridx=2;

                   constraints.gridy=1;

                   constraints.gridwidth=1;

                   constraints.gridheight=1;

                   layout.setConstraints(b3,constraints);

                   add(b3);
```

```
Choice c=new Choice();

c.add("C++");

c.add("JAVA");

constraints.gridx=1;

constraints.gridy=2;

constraints.gridwidth=2;

constraints.gridheight=1;

layout.setConstraints(c,constraints);

add(c);

TextField tf=new TextField("textfield");

constraints.gridx=0;

constraints.gridy=3;

constraints.gridwidth=2;

constraints.gridheight=1;

layout.setConstraints(tf,constraints);

add(tf);

TextArea ta1=new TextArea("welcome",5,9);

constraints.gridx=2;

constraints.gridy=3;

constraints.gridwidth=1;

constraints.gridheight=1;

layout.setConstraints(ta1,constraints);

add(ta1);

setSize(300,300);

setVisible(true);
```

```
        }
        public static void main(String arg[]){
                new GridBagDemo();
        }
}
```



5.1.6 Panel

- The Panel is a simplest container class. It provides space in which an application can attach any other component. It inherits the Container class.
- It doesn't have title bar.
- You can also place panels in a panel.
- FlowLayout is the default layout for panel.

```
    Panel()

    Panel(LayoutManager layout)
```

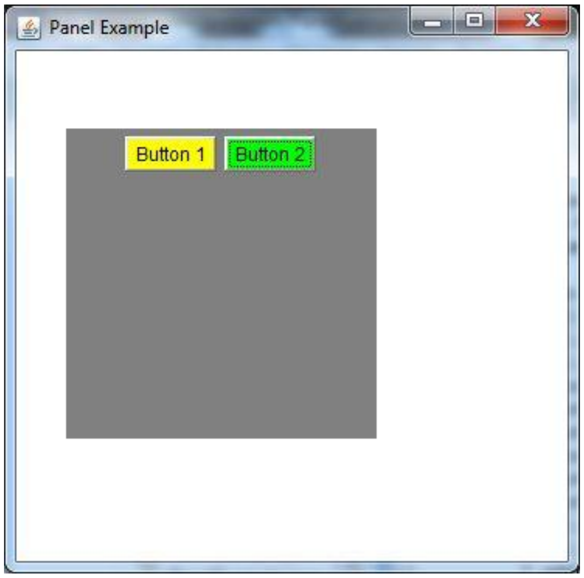
AWT Panel class declaration

public class Panel extends Container implements Accessible

Panel Example

```
import java.awt.*;
public class PanelExample {
    PanelExample()
        {
        Frame f= new Frame("Panel Example");
        Panel panel=new Panel();
        panel.setBounds(40,80,200,200);
        panel.setBackground(Color.gray);
        Button b1=new Button("Button 1");
        b1.setBounds(50,100,80,30);
        b1.setBackground(Color.yellow);
        Button b2=new Button("Button 2");
        b2.setBounds(100,100,80,30);
        b2.setBackground(Color.green);
        panel.add(b1); panel.add(b2);
        f.add(panel);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
        }
        public static void main(String args[])
        {
        new PanelExample();
        }
}
```

Output:



5.1.7 Canvas

The Canvas control represents a blank rectangular area where the application can draw or trap input events from the user. It inherits the Component class.

Constructors

Canvas()

Methods

setSize(int width, int height)

setBackground(Color)

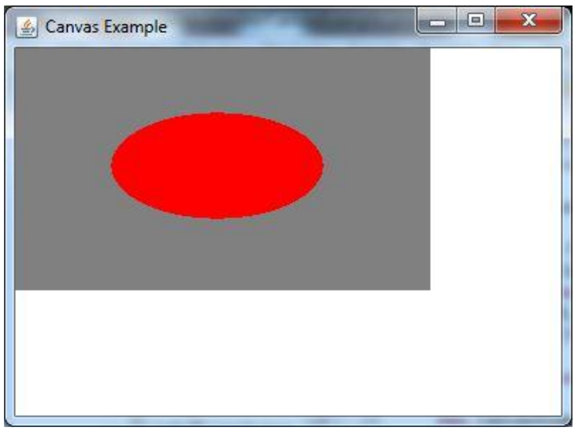
Canvas class Declaration

public class Canvas extends Component implements Accessible

Canvas Example

```
import java.awt.*;
public class CanvasExample
{
  public CanvasExample()
  {
    Frame f= new Frame("Canvas Example");
    f.add(new MyCanvas());
    f.setLayout(null);
    f.setSize(400, 400);
    f.setVisible(true);
  }
  public static void main(String args[])
  {
    new CanvasExample();
  }
}
class MyCanvas extends Canvas
{
        public MyCanvas() {
        setBackground (Color.GRAY);
        setSize(300, 200);
     }
  public void paint(Graphics g)
  {
    g.setColor(Color.red);
    g.fillOval(75, 75, 150, 75);
  }
}
```

Output:



5.1.8 Graphics methods for drawing shapes

g.drawString (str, x, y); //Puts string at x,y

g.drawLine(x1, y1, x2, y2) //Line from x1, y1 to x2, y2

g.drawRect(x1, y1, width, height) //Draws rectangle with upper left corner x1, y1

g.fillRect(x1, y1, width, height) //Draws a solid rectangle.

g.drawRoundRect(x, y, width, height, arcWidth, arcHeight) //Draws rectangle with rounded corners.

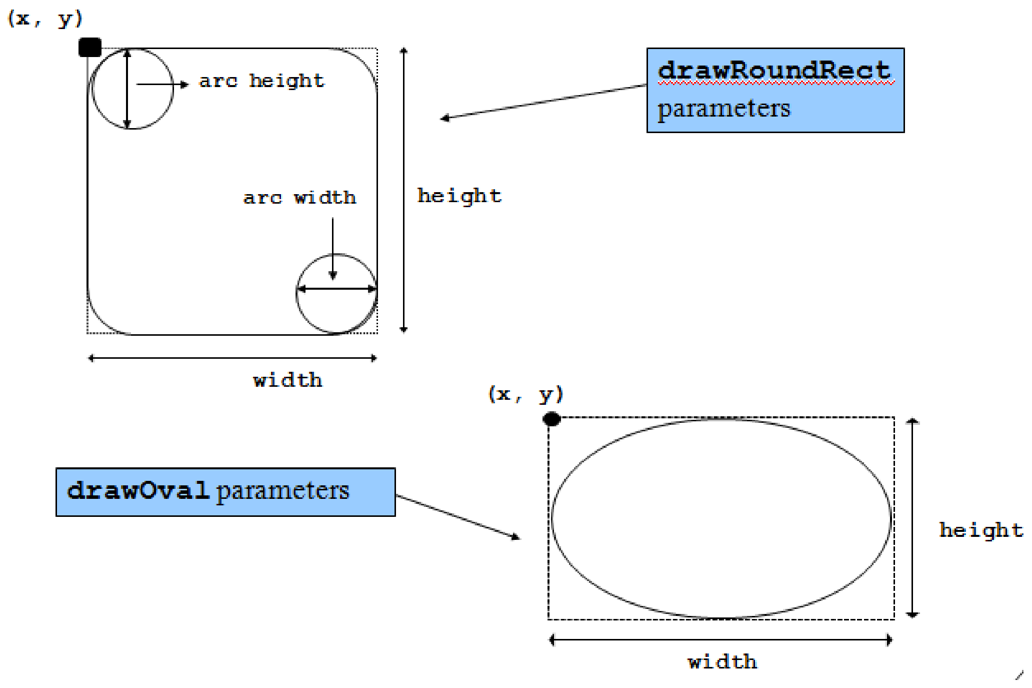
g.fillRoundRect(x, y, width, height, arcWidth, arcHeight) //Draws a solid rectangle with rounded corners.

g.drawOval(x1, y1, width, height) //Draws an oval with specified width and height. The bounding rectangle's top left corner is at the coordinate (x,y).The oval touches all four sides all four sides of the bounding rectangle.

g.fillOval(x1, y1, width, height)　　　　　//Draws a filled oval.

g.setColor(Color.RED)　　　　　　　　//Sets color, it is remain active until new color is set.

g.drawArc(x1, y1, width, height, startAngle,arcAngle)　//Draws an arc relative to the bounding rectangle's top-left coordinates with the specified width and height. The arc segment is drawn starting at startAngle and sweeps arcAngle degrees.



Drawing Polygons and Polylines

- Polygon - multisided shape
- Polyline - series of connected points
- Methods of class Polygon

drawPolygon(xPoints[], yPoints[], points) //Draws a polygon, with x and y points specified in arrays. Last argument specifies number of points

//Closed polygon, even if last point different from first.

drawPolyline (xPoints[], yPoints[],points) //Draws a polyline

Example on Draw lines

```
import java.awt .*;

import java.applet.*;

/*<applet code="Lines" width=300 height=200></applet>*/

public class Lines extends Applet {

        public void paint(Graphics g) {

                g.drawLine(0, 0, 100, 100);

                g.drawLine(0, 100, 100, 0);

                g.drawLine(40, 25, 250, 180);

                g.drawLine(75, 90, 400, 400);

                g.drawLine(20, 150, 400, 40);

                g.drawLine(5, 290, 80, 19);

        }

}
```

Example on Draw Ellipses

```
import java.awt.*;

import java.applet.*;

/*<applet code="Ellipses" width=300 height=200></applet>*/

public class Ellipses extends Applet {

        public void paint(Graphics g) {

                g.drawOval(10, 10, 50, 50);

                g.fillOval(100, 10, 75, 50);
        g.drawOval(190, 10, 90, 30);

        g.fillOval(70, 90, 140, 100);

    }

}
```

Example on Draw Polygon

```
import java.awt.*;

import java.applet.*;

/*<applet code="Hexagon" width=230 height=210></applet>*/

public class Hexagon extends Applet {

    public void paint(Graphics g) {

        int xpoints[] = {20,30,30,20,10,10};

        int ypoints[] = {10,20,30,40,30,20};

        int num = 6;

        g.drawPolygon(xpoints, ypoints, num);

    }

}
```

Example on Draw Arcs

```
import java.awt.*;

import java.applet.*;

/*<applet code="Arcs" width=300 height=200></applet>*/

public class Arcs extends Applet {

    public void paint(Graphics g) {
```

```
		g.drawArc(10, 40, 70, 70, 0, 75);

		g.fillArc(100, 40, 70, 70, 0, 75);

		g.drawArc(10, 100, 70, 80, 0, 175);

		g.fillArc(100, 100, 70, 90, 0, 270);

		g.drawArc(200, 80, 80, 80, 0, 180);

	}

}
```

Example on Draw rectangles

```
import java.awt.*;

import java.applet.*;

/*<applet code="Rectangles" width=300 height=200></applet>*/

public class Rectangles extends Applet {

	public void paint(Graphics g) {

		g.drawRect(10, 10, 60, 50);

		g.fillRect(100, 10, 60, 50);

		g.drawRoundRect(190, 10, 60, 50, 15, 15);

		g.fillRoundRect(70, 90, 140, 100, 30, 40);

	}

}
```

5.2 INTRODUCTION TO SWINGS

JAVA Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

- AWT component set is no longer widely used to create graphical user interfaces.
- One reason for the limited nature of the AWT is that it translates its various visual components into their corresponding, platform-specific equivalents, or *peers*. The look and feel of a component is defined by the platform, not by Java.
- Because the AWT components use native code resources, they are referred to as *heavyweight*.
- Most programmers use Swing for this purpose.Swing is a framework that provides more powerful and flexible GUI components than AWT.
- Swing eliminates a number of the limitations inherent in the AWT.Swing is built on the foundation of the AWT. This is why the AWT is still a crucial part of Java.
- Swing also uses the same event handling mechanism as the AWT
- Two key features: lightweight components and a pluggable look and feel.
- Swing components are *lightweight*. i.e they are written entirely in Java and do not map directly to platform-specific peers. lightweight components are more efficient and more flexible. The look and feel of each component is determined by Swing, not by the underlying operating system.
- Each component will work in a consistent manner across all platforms.
- A Swing GUI consists of two key items: *components* and *containers*
- A *component* is an independent visual control, such as a push button or slider.
- A *container* holds a group of components
- **JFrame**, **JApplet**,**JWindow**, and **Jdialog** containers do not inherit **JComponent**

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

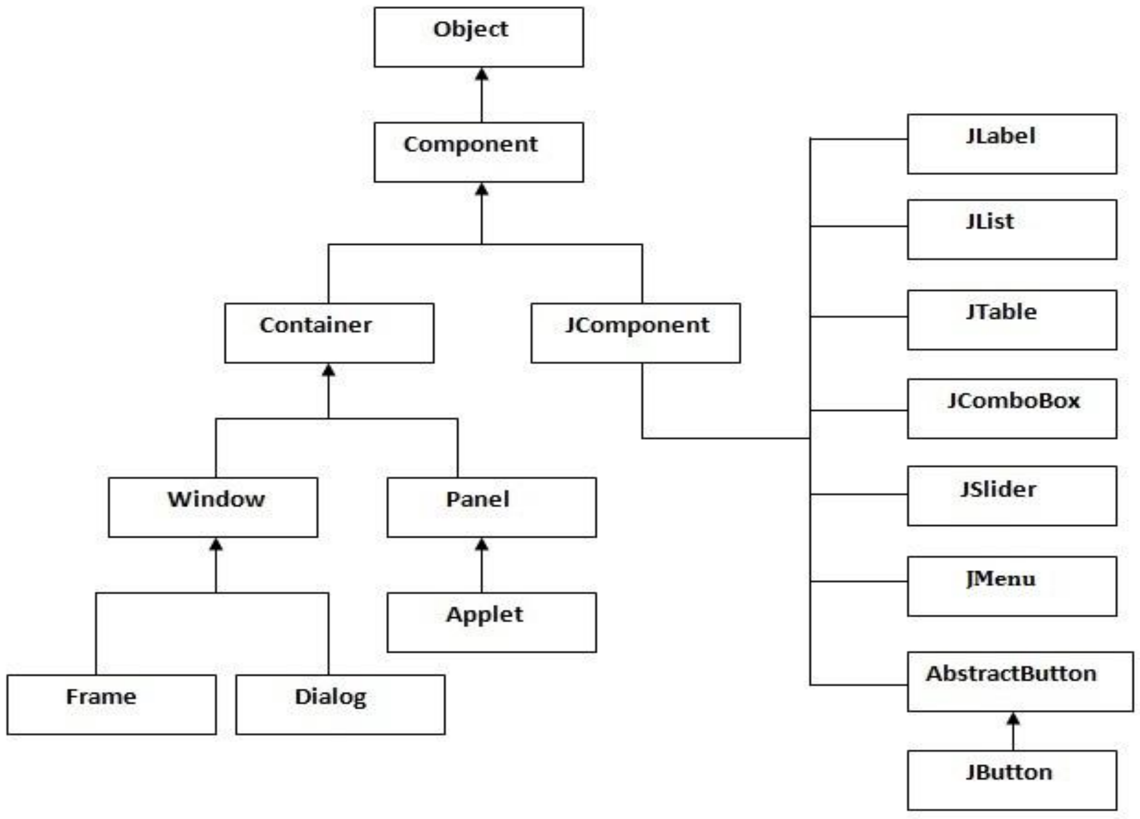
5.2.1 Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

	Java AWT	**Java Swing**
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look**	Swing **supports pluggable look and feel**.

	and feel.	
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

The hierarchy of java swing API is given below

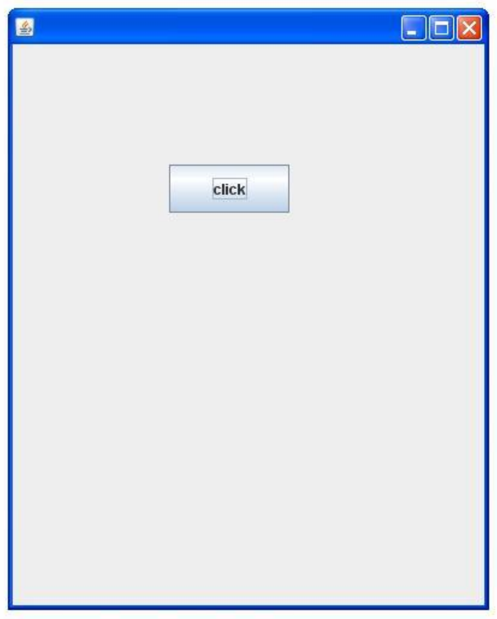


Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.

public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Simple Java Swing Example

```
import javax.swing.*;
public class FirstSwingExample {
public static void main(String[] args) {
JFrame f=new JFrame();//creating instance of JFrame
JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);//x axis, y axis, width, height
f.add(b);//adding button in JFrame
f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
}
```



Example of Swing by Association inside constructor

```
import javax.swing.*;
public class Simple {
JFrame f;
Simple(){
```

```
f=new JFrame();//creating instance of JFrame
JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);
f.add(b);//adding button in JFrame
f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
public static void main(String[] args) {
    new Simple();
}
}
```

The setBounds(int xaxis, int yaxis, int width, int height)is used in the above example that sets the position of the button.

Example of Swing by inheritance

```
import javax.swing.*;
public class Simple2 extends JFrame{//inheriting JFrame
JFrame f;
Simple2(){
JButton b=new JButton("click");//create button
b.setBounds(130,100,100, 40);
add(b);//adding button on frame
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main(String[] args) {
new Simple2();
}}
```

5.2.2 JButton

- The JButton class provides the functionality of a push button
- JButton allows an icon, a string, or both to be associated with the push button.
- Constructors:
- JButton(Icon *icon*)
- JButton(String *str*)
- JButton(String *str*, Icon *icon*)

Here, *str* and *icon* are the string and icon used for the button.

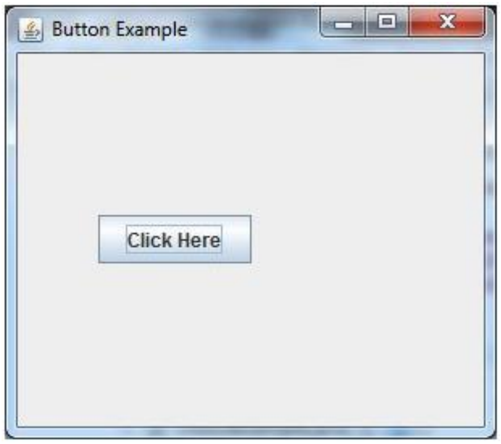
Let's see the declaration for javax.swing.JButton class.

    public class JButton extends AbstractButton implements Accessible

Example

```
import javax.swing.*;
public class ButtonExample {
public static void main(String[] args) {
JFrame f=new JFrame("Button Example");
JButton b=new JButton("Click Here");
  b.setBounds(50,100,95,30);
  f.add(b);
  f.setSize(400,400);
  f.setLayout(null);
 f.setVisible(true);
}
}
```

Output:



5.2.3 JTextField

- JTextField is the simplest Swing text component.
- It is also probably its most widely used text component.
- JTextField allows to edit one line of text
- Constructors:

    JTextField(int *cols*)
    JTextField(String *str*, int *cols*)
    JTextField(String *str*)

Here, *str* is the string to be initially presented, and *cols* is the number of columns in the textfield.

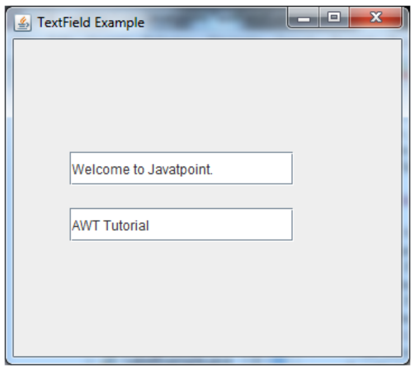
- If no string is specified, the text field is initially empty.
- If the number of columns is not specified, the text field is sized to fit the specified string.

Declaration for javax.swing.JTextField class.
public class JTextField extends JTextComponent implements SwingConstants

Example

```
import javax.swing.*;
class TextFieldExample
{
public static void main(String args[])
    {
      JFrame f= new JFrame("TextField Example");
       JTextField t1,t2;
       t1=new JTextField("Welcome to Javatpoint.");
       t1.setBounds(50,100, 200,30);
       t2=new JTextField("AWT Tutorial");
       t2.setBounds(50,150, 200,30);
       f.add(t1); f.add(t2);
       f.setSize(400,400);
       f.setLayout(null);
       f.setVisible(true);
       }
       }
```

Output:



Welcome to Javatpoint.

AWT Tutorial

5.2.4 JCheckBox

Constructor:

- JCheckBox(String *str*)

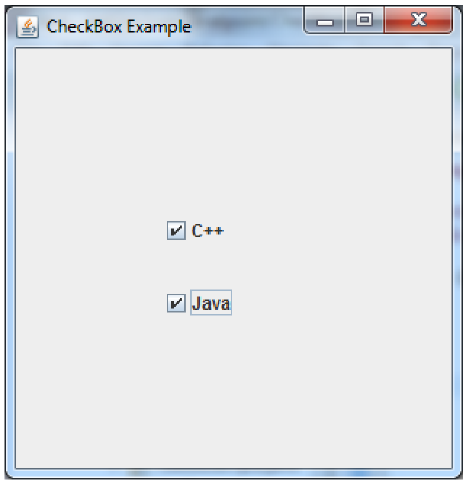
- It creates a check box that has the text specified by *str* as a label
- When the user selects or deselects a check box, an ItemEvent is generated

Declaration for javax.swing.JCheckBox class.
public class JCheckBox extends JToggleButton implements

Example

```
import javax.swing.*;
public class CheckBoxExample
{
     CheckBoxExample(){
        JFrame f= new JFrame("CheckBox Example");
        JCheckBox checkBox1 = new JCheckBox("C++");
        checkBox1.setBounds(100,100, 50,50);
        JCheckBox checkBox2 = new JCheckBox("Java", true);
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1);
        f.add(checkBox2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
     }
public static void main(String args[])
    {
    new CheckBoxExample();
    }}
```

Output:



5.2.5 JRadioButton

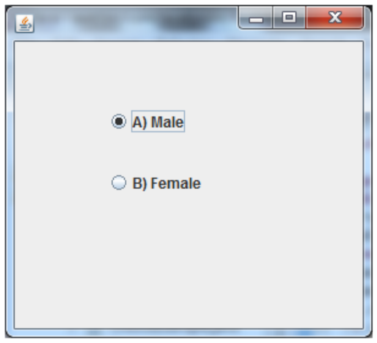
- Radio buttons are a group of mutually exclusive buttons, in which only one button can be selected at any one time.
- They are supported by the JRadioButton class
- JRadioButton(String *str*)
    - Here, *str* is the label for the button
- JRadioButton generates action events, item events, and change events each time the button selection changes.

Declaration for javax.swing.JRadioButton class.
public class JRadioButton extends JToggleButton implements Accessible

Example

```
import javax.swing.*;
public class RadioButtonExample {
Frame f;
RadioButtonExample(){
f=new JFrame();
JRadioButton r1=new JRadioButton("A) Male");
JRadioButton r2=new JRadioButton("B) Female");
r1.setBounds(75,50,100,30);
r2.setBounds(75,100,100,30);
ButtonGroup bg=new ButtonGroup();
bg.add(r1);bg.add(r2);
f.add(r1);f.add(r2);     f.setSize(300,300);    f.setLayout(null);    f.setVisible(true);
}
public static void main(String[] args) {
  new RadioButtonExample();
}    }
```

Output:



5.2.6 JList

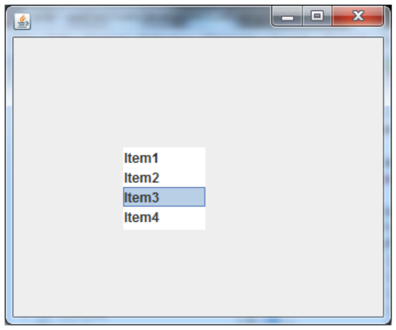
- The basic list class is called JList.
- It supports the selection of one or more items from a list.
- Although the list often consists of strings, it is possible to create a list of just about any object that can be displayed
- Constructor:
    - JList(E[ ] *items*)
    - E represents the type of the items in the list.

Declaration for javax.swing.JList class.
public class JList extends JComponent implements Scrollable, Accessible

Example

```
import javax.swing.*;
public class ListExample
{
    ListExample(){
      JFrame f= new JFrame();
      DefaultListModel<String> l1 = new DefaultListModel<>();
        l1.addElement("Item1");
        l1.addElement("Item2");
        l1.addElement("Item3");
        l1.addElement("Item4");
        JList<String> list = new JList<>(l1);
        list.setBounds(100,100, 75,75);
        f.add(list);        f.setSize(400,400);        f.setLayout(null);        f.setVisible(true)
    }
public static void main(String args[])    {
  new ListExample();
   }}
```

Output:



5.2.7 JTextArea

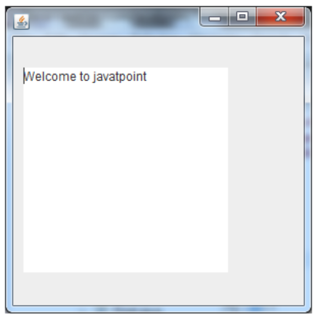
- A JTextArea is a multi-line area that displays plain text
- Constructor:
    - JTextArea(): Creates a text area that displays no text initially.
    - JTextArea(String s): Creates a text area that displays specified text initially.
    - JTextArea(int row, int column): Creates a text area with the specified number of rows and columns that displays no text initially.
    - JTextArea(String s, int row, int column): Creates a text area with the specified number of rows and columns that displays specified text.

Declaration for javax.swing.JTextArea class.
public class JTextArea extends JTextComponent

Example

```
import javax.swing.*;
public class TextAreaExample
{
    TextAreaExample(){
      JFrame f= new JFrame();
      JTextArea area=new JTextArea("Welcome to javatpoint");
      area.setBounds(10,30, 200,200);
      f.add(area);
      f.setSize(300,300);
      f.setLayout(null);
      f.setVisible(true);
    }
public static void main(String args[])
   {
  new TextAreaExample();
   }}
```

Output:



```
Welcome to javatpoint
```

5.3 INTRODUCTION TO NETWORKING

5.3.1 Java Networking

The term network programming refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.

The java.net package contains a collection of classes and interfaces which are used to implement programs to communicate across network.

The java.net provides support for the two common network protocols:

TCP: Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.

UDP: User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between

IP Address (Internet Protocol Address)

It is an unique identifier which is assigned to device (computer) to identify itself and communicate with other devices in the Internet Protocol network. Networks using the TCP/IP protocol route messages based on the IP address of the destination.

The format of an IP address is a 32-bit numeric address written as four numbers separated by periods. Each number can be 0 to 255.

For example: 1.160.10.240 could be an IP address.

Otherwise, IP Address is used to identify host.

Port

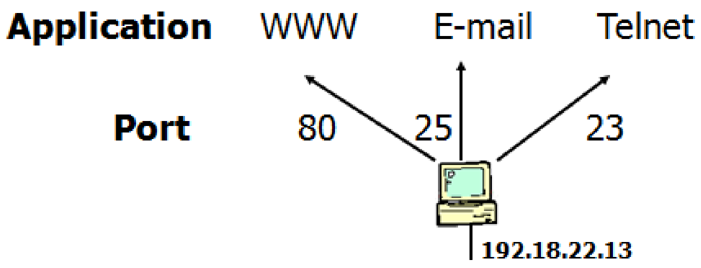
Generally a host has many applications.

Port is used to identify the specific application .

It is a 16-bit identifier.

A port represents an endpoint for network communications.

Port numbers allow different applications on the same computer to utilize network resources without interfering with each other.



5.3.2 InetAddress

The InetAddress class is used to encapsulate both the numerical IP address and the domain name for that address. You interact with this class by using the name of an IP host, which is more convenient and understandable than its IP address. The InetAddress class hides the number inside. InetAddress can handle both IPv4 and IPv6 addresses.

Factory Methods

The InetAddress class has no visible constructors. To create an InetAddress object, you have to use one of the available factory methods. *Factory methods* are merely a convention whereby static methods in a class return an instance of that class. This is done in lieu of overloading a constructor with various parameter lists when having unique method names makes the results much clearer. Three commonly used InetAddress factory methods are shown here:

static InetAddress getLocalHost( ) throws UnknownHostException

static InetAddress getByName(String *hostName*) throws UnknownHostException

static InetAddress[ ] getAllByName(String *hostName*) throws UnknownHostException

The getLocalHost( ) method simply returns the InetAddress object that represents the local host. The getByName( ) method returns an InetAddress for a host name passed to it. If these methods are unable to resolve the host name, they throw an UnknownHostException. The getAllByName( ) factory method returns an array of InetAddresses that represent all of the addresses that a particular name resolves to. It will also throw an UnknownHostException if it can't resolve the name to at least one address.

InetAddress also includes the factory method getByAddress( ), which takes an IP address and returns an InetAddress object. Either an IPv4 or an IPv6 address can be used.

Example to print the addresses and names of the local machine and two Internet web sites:

```
// Demonstrate InetAddress.

import java.net.*;

class InetAddressTest

{

public static void main(String args[]) throws UnknownHostException {

InetAddress Address = InetAddress.getLocalHost();

System.out.println(Address);

Address = InetAddress.getByName("www.HerbSchildt.com");

System.out.println(Address);

InetAddress SW[] = InetAddress.getAllByName("www.nba.com");

for (int i=0; i<SW.length; i++)

System.out.println(SW[i]);

}

}
```

Output:

```
default/166.203.115.212

www.HerbSchildt.com/216.92.65.4

www.nba.com/216.66.31.161

www.nba.com/216.66.31.179
```

Instance Methods

The InetAddress class has several other methods, which can be used on the objects returned by the methods just discussed. Here are some of the more commonly used methods:

boolean equals(Object *other*) Returns true if this object has the same Internet address as *other*. byte[ ]

getAddress( ) Returns a byte array that represents the object's IP address in network byte order.

String getHostAddress( ) Returns a string that represents the host address associated with the InetAddress object.

String getHostName( ) Returns a string that represents the host name associated with the InetAddress object.

boolean isMulticastAddress( ) Returns true if this address is a multicast address. Otherwise, it returns false.

String toString( ) Returns a string that lists the host name and the IP address for convenience.

Internet addresses are looked up in a series of hierarchically cached servers. That means that your local computer might know a particular name-to-IP-address mapping automatically, such as for itself and nearby servers. For other names, it may ask a local DNS server for IP address information. If that server doesn't have a particular address, it can go to a remote site and ask for it. This can continue all the way up to the root server.

Inet4Address and Inet6Address

Java includes support for both IPv4 and IPv6 addresses. Because of this, two subclasses ofInetAddress were created: Inet4Address and Inet6Address. Inet4Address represents atraditional-style IPv4 address. Inet6Address encapsulates a newer IPv6 address. Because theyare subclasses of InetAddress, an InetAddress reference can refer to either. This is one way that Java was able to add IPv6 functionality without breaking existing code or adding many more classes. For the most part, you can simply use InetAddress when working with IP addresses because it can accommodate both styles.

5.3.3 Socket Class

A socket represents a single connection between two network applications. It is an object used for network programming.
Sockets are bidirectional, meaning that either side of the connection is capable of both sending and receiving data.

A socket is bound to a specific port number.
Network communication using Sockets is very much similar to performing file I/O.
The streams used in file I/O operation are also applicable to socket-based I/O.

5.3.3.1 ServerSocket Constructors

public ServerSocket() throws IOException
Creates an unbound server socket. When using this constructor, use the bind() method when you are ready to bind the server socket.

public ServerSocket(int port) throws IOException
Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application**.**

5.3.3.2 ServerSocket Methods

public int getLocalPort()
Returns the port that the server socket is listening on.

public Socket accept() throws IOException
Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out.

public void bind(SocketAddress host, int port)
Binds the socket to the specified server and port in the SocketAddress object. Use this method if you instantiated the ServerSocket using the no-argument constructor.

5.3.3.3 Socket Constructors

public Socket()
Creates an unconnected socket. Use the connect() method to connect this socket to a server.

public Socket(String host, int port) throws UnknownHostException, IOException.
It attempts to connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server.

5.3.3.5 Socket Methods

public void connect(SocketAddress host, int port) throws IOException
It connects the socket to the specified host. This method is needed only when you instantiated the Socket using the no-argument constructor.

public InputStream getInputStream() throws IOException
Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.

public OutputStream getOutputStream() throws IOException
Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket.

public void close() throws IOException
Closes the socket, which makes this Socket object no longer capable of connecting again to any server.

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Example of Java Socket Programming

MyServer.java

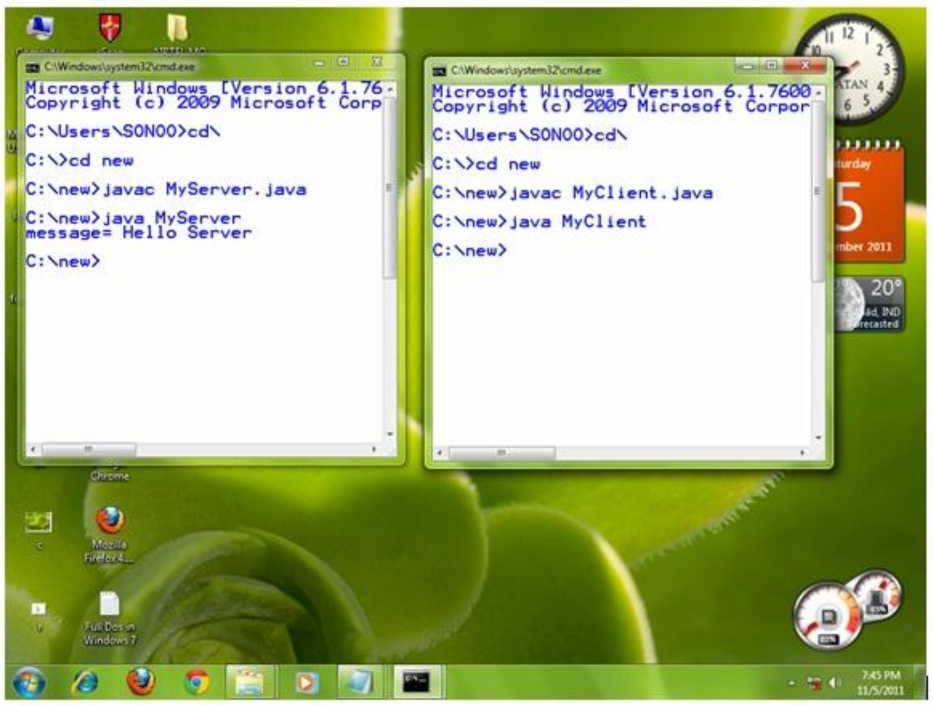
```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(6666);
ocket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e){System.out.println(e);}
}
}
```

MyClient.java

```
import java.io.*;
import java.net.*;
public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```

To execute this program open two command prompts and execute each program at each command prompt as displayed in the below figure.

After running the client application, a message will be displayed on the server console.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.76
Copyright (c) 2009 Microsoft Corp

C:\Users\SONOO>cd\

C:\>cd new

C:\new>javac MyServer.java

C:\new>java MyServer
message= Hello Server

C:\new>
```

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600
Copyright (c) 2009 Microsoft Corpor

C:\Users\SONOO>cd\

C:\>cd new

C:\new>javac MyClient.java

C:\new>java MyClient

C:\new>
```

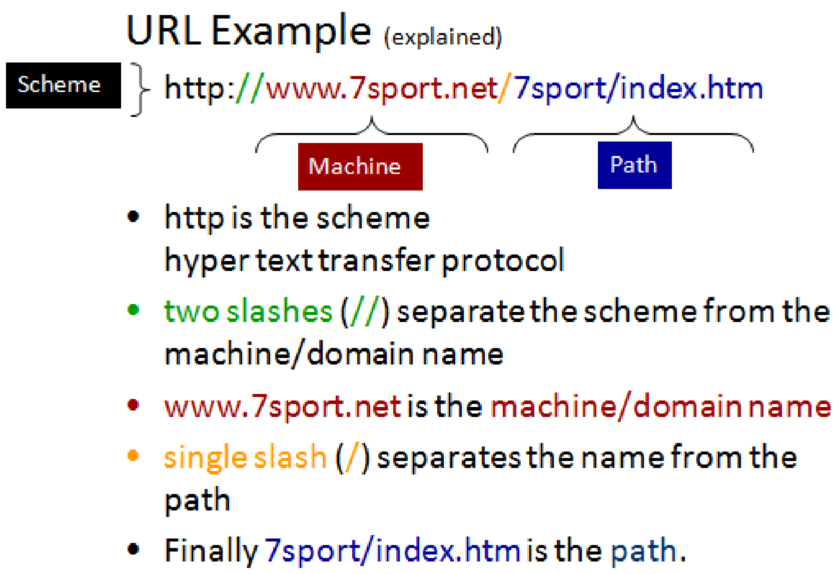
5.3.4 URL Class

A Uniform Resource Locator (URL) is a standard way developed to specify the location of a resource available electronically.The Java URL class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web.

A URL contains many information:

1. Protocol
2. Server name or IP Address

3. Port Number
4. File Name or directory name



- http is the scheme
  hyper text transfer protocol
- two slashes (//) separate the scheme from the machine/domain name
- www.7sport.net is the machine/domain name
- single slash (/) separates the name from the path
- Finally 7sport/index.htm is the path.

One commonly used form specifies the URL with a string that is identical to what you seedisplayed in a browser:

URL(String urlSpecifier) throws MalformedURLException

The next two forms of the constructor allow you to break up the URL into its component parts:

URL(String protocolName, String hostName, int port, String path)throws MalformedURLException

URL(String protocolName, String hostName, String path) throws MalformedURLException

Another frequently used constructor allows you to use an existing URL as a reference context and then create a new URL from that context. Although this sounds a little contorted, it’s really quite easy and useful.

URL(URL urlObj, String urlSpecifier) throws MalformedURLException

Commonly used methods of Java URL class

The java.net.URL class provides many methods. The important methods of URL class are given below.

Method	**Description**
public String getProtocol()	it returns the protocol of the URL.
public String getHost()	it returns the host name of the URL.
public String getPort()	it returns the Port Number of the URL.
public String getFile()	it returns the file name of the URL.
Public URLConnection openConnection()	it returns the instance of URLConnection i.e. associated with this URL.

Example on URL Class

```
// Demonstrate URL.

import java.net.*;

class URLDemo {

public static void main(String args[]) throws MalformedURLException {

URL hp = new URL(http://www.HerbSchildt.com/WhatsNew");

System.out.println("Protocol: " + hp.getProtocol());

System.out.println("Port: " + hp.getPort());

System.out.println("Host: " + hp.getHost());

System.out.println("File: " + hp.getFile());

System.out.println("Ext:" + hp.toExternalForm());

}

}
```

Output:

```
Protocol: http

Port: -1

Host: www.HerbSchildt.com

File: /WhatsNew
```

Ext:http://www.HerbSchildt.com/WhatsNew

Notice that the port is –1; this means that a port was not explicitly set. Given a URL object, you can retrieve the data associated with it.

To access the actual bits or content information of a URL, create a URLConnection object from it, using its openConnection( ) method,like this:

urlc = url.openConnection()

openConnection( ) has the following general form:

URLConnection openConnection( ) throws IOException

It returns a URLConnection object associated with the invoking URL object. Notice that it may throw an IOException.