GUI Programming with Swing

Event Handling, SWING Application AND APPLETS

5.1 GUI Programming with Swings

5.1.1 Introduction

Computer users today expect to interact with their computers using a graphical user interface (GUI). Java can be used to write GUI programs ranging from simple applets which run on a Web page to sophisticated stand-alone applications.

GUI programs differ from traditional "straight-through" programs . One big difference is that GUI programs are event-driven. That is, user actions such as clicking on a button or pressing a key on the keyboard generate events, and the program must respond to these events as they occur.

And of course, objects are everywhere in GUI programming. Events are objects. Colors and fonts are objects. GUI components such as buttons and menus are objects. Events are handled by instance methods contained in objects. In Java, GUI programming is object-oriented programming.

The Basic GUI Application

There are two basic types of GUI program in Java: ***stand-alone applications and applets.*** An applet is a program that runs in a rectangular area on a Web page. very complex.

A stand-alone application is a program that runs on its own, without depending on a Web browser. You've been writing stand-alone applications all along. Any class that has a main() routine defines a stand-alone application; running the program just means executing this main() routine.

A GUI program offers a much richer type of user interface, where the user uses a mouse and keyboard to interact with GUI components such as windows, menus, buttons, check boxes, text input boxes, scroll bars, and so on. The main routine of a GUI program creates one or more such components and displays them on the computer screen. Very often, that's all it does. Once a GUI component has been created, it follows its own programming—programming that tells it how to draw itself on the screen and how to respond to events such as being clicked on by the user.

A GUI program doesn't have to be immensely complex. We can, for example, write a very simple GUI "Hello World" program that says "Hello" to the user, but does it by opening a window where the the greeting is displayed:

```
import javax.swing.JOptionPane;

    public class HelloWorldGUI1 {

    public static void main(String[] args) {

    JOptionPane.showMessageDialog( null, "Hello World!" );

    }

}
```

When this program is run, a window appears on the screen that contains the message "Hello World!". The window also contains an "OK" button for the user to click after reading the message. When the user clicks this button, the window closes and the program ends. By the way, this program can be placed in a file named HelloWorldGUI1.java, compiled, and run just like any other Java program.

5.1.2 Limitations of AWT

The AWT defines a basic set of controls, windows, and dialog boxes that support a usable, but limited graphical interface. 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. This means that 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 heavy weight.The use of native peers led to several problems. First, because of variations between operating systems, a component might look, or even act, differently on different

platforms. This variability threatened java's philosophy: write once, run anywhere.Second, the look and feel of each component was fixed and could not be changed. Third,the use of heavyweight components caused some frustrating restrictions.

Summary on limitations of AWT

AWT supports limited number of GUI components

AWT component are Heavy weight components

AWT components are developed by using platform specific code

AWT components behaves differently in different Operating Systems.

AWT components is converted by the native code of the Operating System.

5.1.3 MVC Architecture

In real time applications, in the case of server side programming one must follow the architecture to develop a distributed application.To develop any distributed application, it is always recommended to follow either 3-tier architecture or 2-tier architecture or n-tier architecture.

3-tier architecture is also known as MVC architecture.

M stands for Model (database programming),
V stands for View (client side programming, HTML/AWT/APPLET/Swing/JSP) and
C stands for Controller (server side programming, Servlets).

Model :

- This is the data layer which consists of the business logic of the system.
- It consists of all the data of the application
- It also represents the state of the application.
- It consists of classes which have the connection to the database.
- The controller connects with model and fetches the data and sends to the view layer.
- The model connects with the database as well and stores the data into a database which is connected to it.

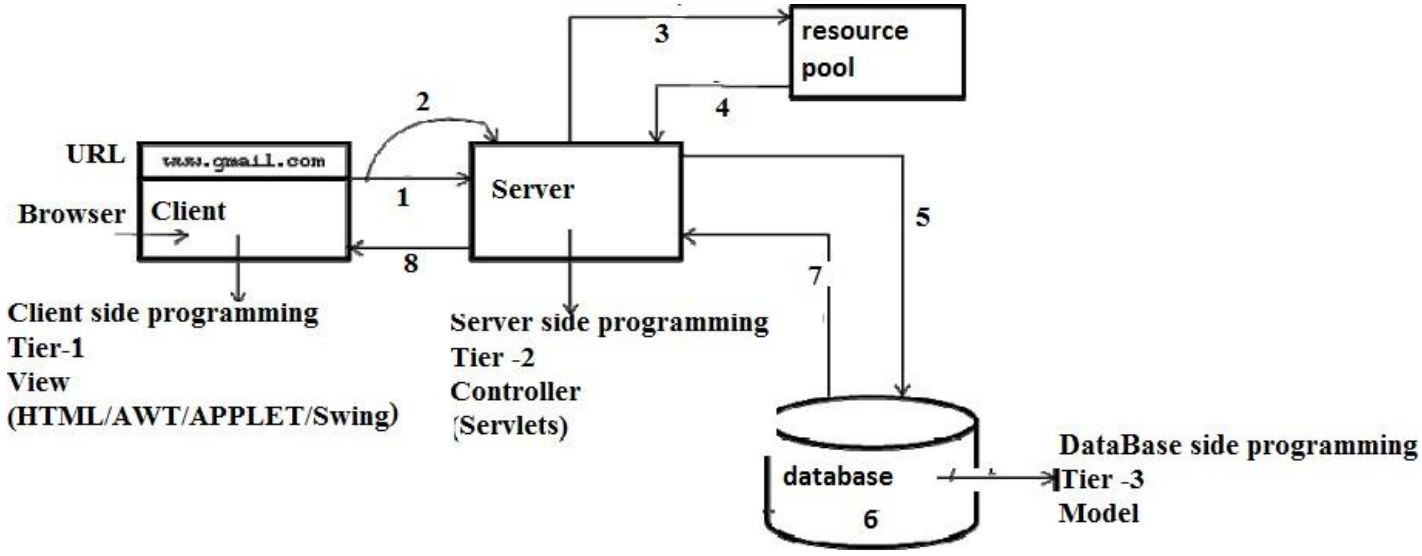
View :

- This is a presentation layer.
-
- It consists of HTML, JSP, etc. into it.
- It normally presents the UI of the application.
- It is used to display the data which is fetched from the controller which in turn fetching data from model layer classes.
- This view layer shows the data on UI of the application.

Controller:

- It acts as an interface between View and Model.
- It intercepts all the requests which are coming from the view layer.
- It receives the requests from the view layer and processes the requests and does the necessary validation for the request.
- This requests is further sent to model layer for data processing, and once the request is processed, it sends back to the controller with required information and displayed accordingly by the view.

The general architecture of MVC or 3-tier:



1. Client makes a request.

2. Server side program receives the request.

3. The server looks for or search for the appropriate resource in the resource pool.

4. If the resource is not available server side program displays a user friendly message (page cannot be displayed). If the resource is available, that program will execute gives its result to server, server interns gives response to that client who makes a request.

5. When server want to deals with database to retrieve the data, server side program sends a request to the appropriate database.

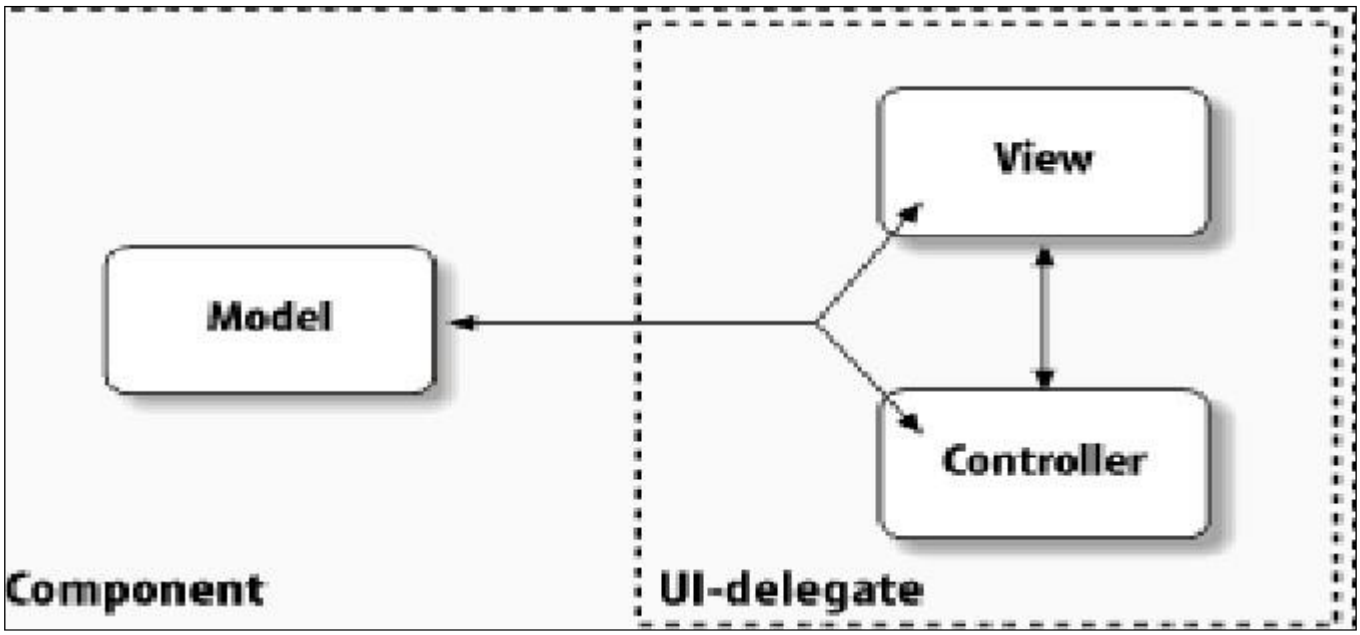
6. Database server receives the server request and executes that request.

7. The database server sends the result back to server side program for further processing.

8. The server side program is always gives response to ‘n’ number of clients concurrently**.**

MVC in Swing

Swing actually uses a simplified variant of the MVC design called the model-delegate . This design combines the view and the controller object into a single element, the UI delegate , which draws the component to the screen and handles GUI events. Bundling graphics capabilities and event handling is somewhat easy in Java, since much of the event handling is taken care of in AWT. As you might expect, the communication between the model and the UI delegate then becomes a two-way street, as shown in Figure below.

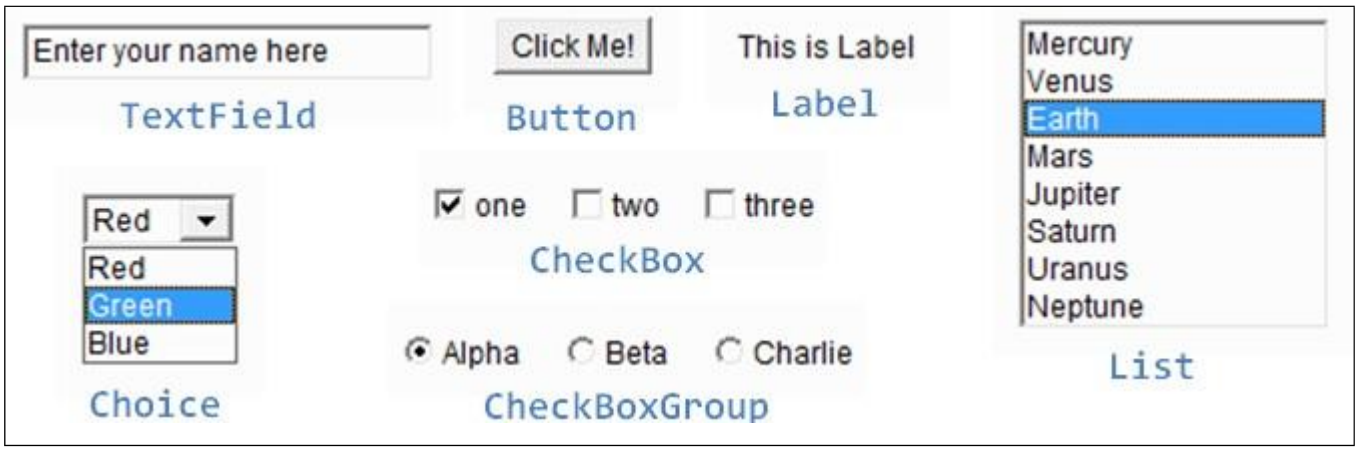


So let's review: each Swing component contains a model and a UI delegate. The model is responsible for maintaining information about the component's state. The UI delegate is responsible for maintaining information about how to draw the component on the screen. In addition, the UI delegate (in conjunction with AWT) reacts to various events that propagate through the component.

Note that the separation of the model and the UI delegate in the MVC design is extremely advantageous. One unique aspect of the MVC architecture is the ability to tie multiple views to a single model. For example, if you want to display the same data in a pie chart and in a table, you can base the views of two components on a single data model. That way, if the data needs to be changed, you can do so in only one place—the views update themselves accordingly

5.1.4 Components

Component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. For examples buttons, checkboxes, list and scrollbars of a graphical user interface.



A Component is an abstract super class for GUI controls and it represents an object with graphical representation.

Every AWT controls inherits properties from Component class

Component	Description
Label	The easiest control to use is a label. A label is an object of type Label, and it contains a string,which it displays. Labels are passive controls that do not support any interaction with theuser. Label defines the following constructors
Button	This class creates a labeled button.
Check Box	A check box is a graphical component that can be in either an on (true) or off (false) state.
Check Box Group	The CheckboxGroup class is used to group the set of checkbox.
List	The List component presents the user with a scrolling list of text items.
Text Field	A TextField object is a text component that allows for the editing of a single line of text.
Text Area	A TextArea object is a text component that allows for the editing of a multiple lines of text.
Choice	A Choice control is used to show pop up menu of choices. Selected choice is shown on the top of the menu.
Canvas	A Canvas control represents a rectangular area where application can draw something or can receive inputs created by user.
Image	An Image control is superclass for all image classes representing graphical images.
Scroll Bar	A Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
Dialog	A Dialog control represents a top-level window with a title and a border used to take

	some form of input from the user.
File Dialog	A FileDialog control represents a dialog window from which the user can select a file.

Commonly used Methods of Component class:

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(intwidth,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.
void remove(Component obj)	Here, obj is a reference to the control you want to remove.
void removeAll( ).	You can remove all controls by

```
Import java.awt.*;

Class swap
{
  Public ststic void main(String argv[])
  {
     Frame f =new Frame("my frame"); \\ Frame( ) Frame(String title)

          Lable l1= new Lable("first");     \\ Lable(  ) Lable( String text) Lable( String text, int alignment)

          Lable l2= new lable("second");    \\Lable.Left, Lable.center, Lable. Right
```

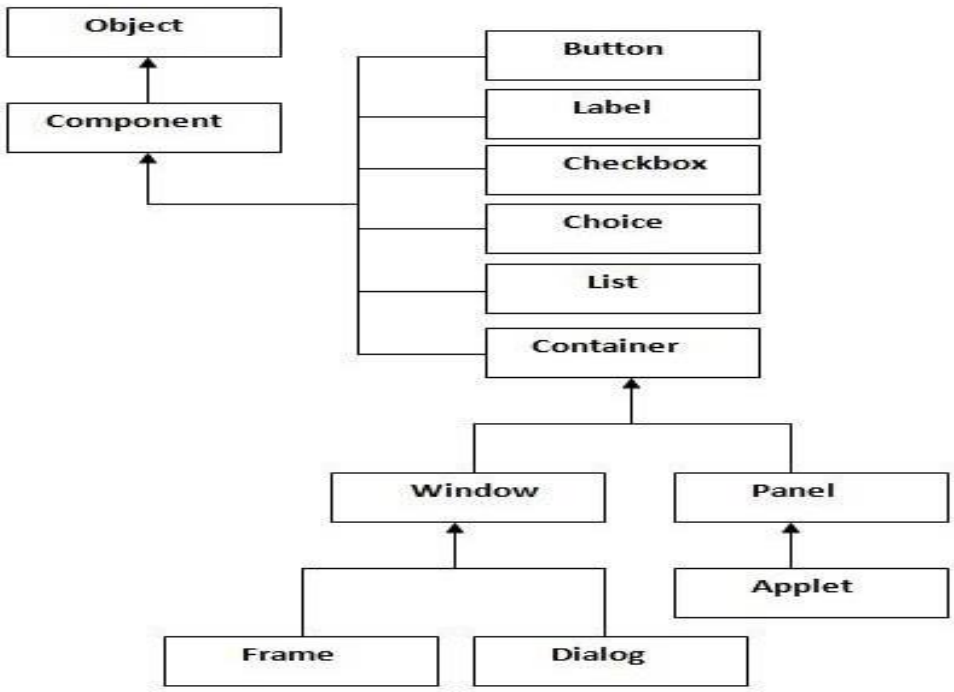
TextField t1= new TextField(10); \\ TextField( ), TextField(int columns), TextField(String text)

TextField t2= new TextField(10); \\ TextField(String text t, in tint column)

Button b= new Button("ok"); \\ Button( ), Button(String text)

```
f.add(l1);
f,add(l2);
f,add(t1);
f,add(t2);
f,add(b);
```

f.setSize(300,300); //component- container-window-Frame
f.setVisible(true);
}



Labels

The easiest control to use is a label. A label is an object of type Label, and it contains a string, which it displays. Labels are passive controls that do not support any interaction with the user.
Label defines the following constructors:

- Label( ) throws HeadlessException
- Label(String str) throws HeadlessException
- **Label(String str, int how)** throws

HeadlessExceptionThe first version creates a blank label.

The second version creates a label that contains the string specified by str. This string is left-justified.

The third version creates a label that contains the string specified by str using the alignment specified by how. The value of how must be one of these three constants: Label.LEFT, Label.RIGHT, or Label.CENTER.

You can set or change the text in a label by using the setText( ) method. You can obtain the current label by calling getText( ).

These methods are shown here:

void setText(String str)

String getText( )

For setText( ), str specifies the new label. For getText( ), the current label is returned.

The below figure represents the appears of Label

Buttons

Perhaps the most widely used control is the push button. A push button is a component that contains a label and that generates an event when it is pressed. Push buttons are objects of type Button.

Button defines these two constructors:

- Button( ) throws HeadlessException
- Button(String str) throws HeadlessException

The first version creates an empty button.

The second creates a button that contains str as a label.

After a button has been created, you can set its label by calling **setLabel( ).** You can retrieve its label by calling **getLabel( ).**

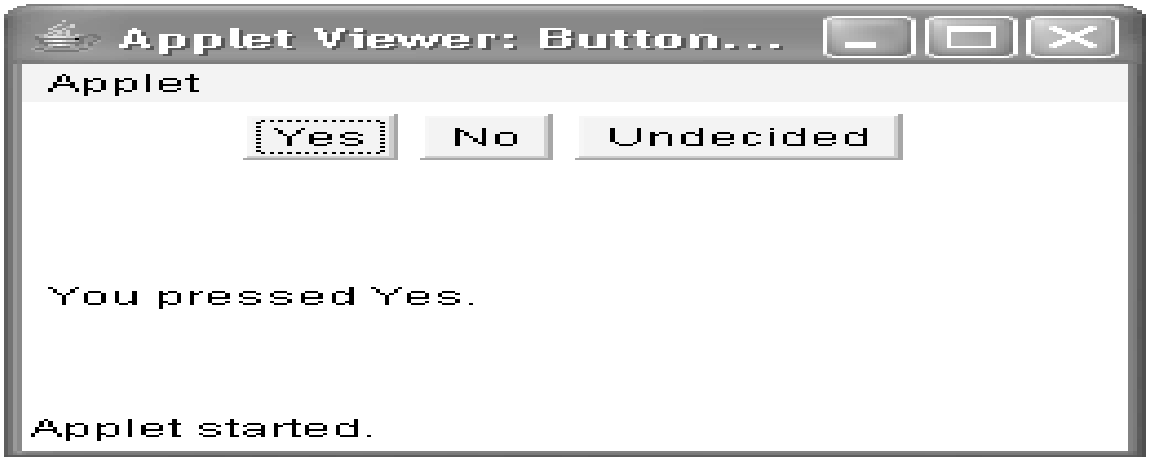
These methods are as follows:

void setLabel(String tr)

String getLabel( )

Here, str becomes the new label for the button

The below figure represents the appears of Buttons



Button b= new Button(“click here”);

b.addActionListener(new MyListener());

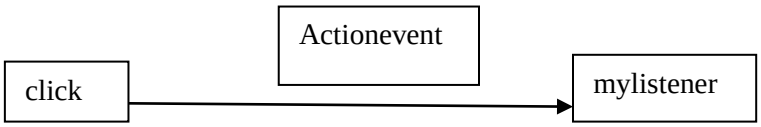
ex:

```
 class Myclass implements ActionListener
{
   Public void actionPerformed(ActionEvent ae)

  {


 }
}
```

```
Import java.awt.*;

Class swap
{
   Public ststic void main(String argv[])
  {
       Frame f =new Frame(“my frame”); \\ Frame( ) Frame(String title)


           Lable l1= new Lable(“first”);      \\ Lable(  ) Lable( String text) Lable( String text, int alignment)

            Lable l2= new lable(“second”, Lable.right);    \\Lable.Left, Lable.center, Lable.
Right


       TextField t1= new TextField(10); \\ TextField( ), TextField(int columns),   TextField(String text)

       TextField t2= new TextField(10); \\ TextField(String text t, in tint column)
```

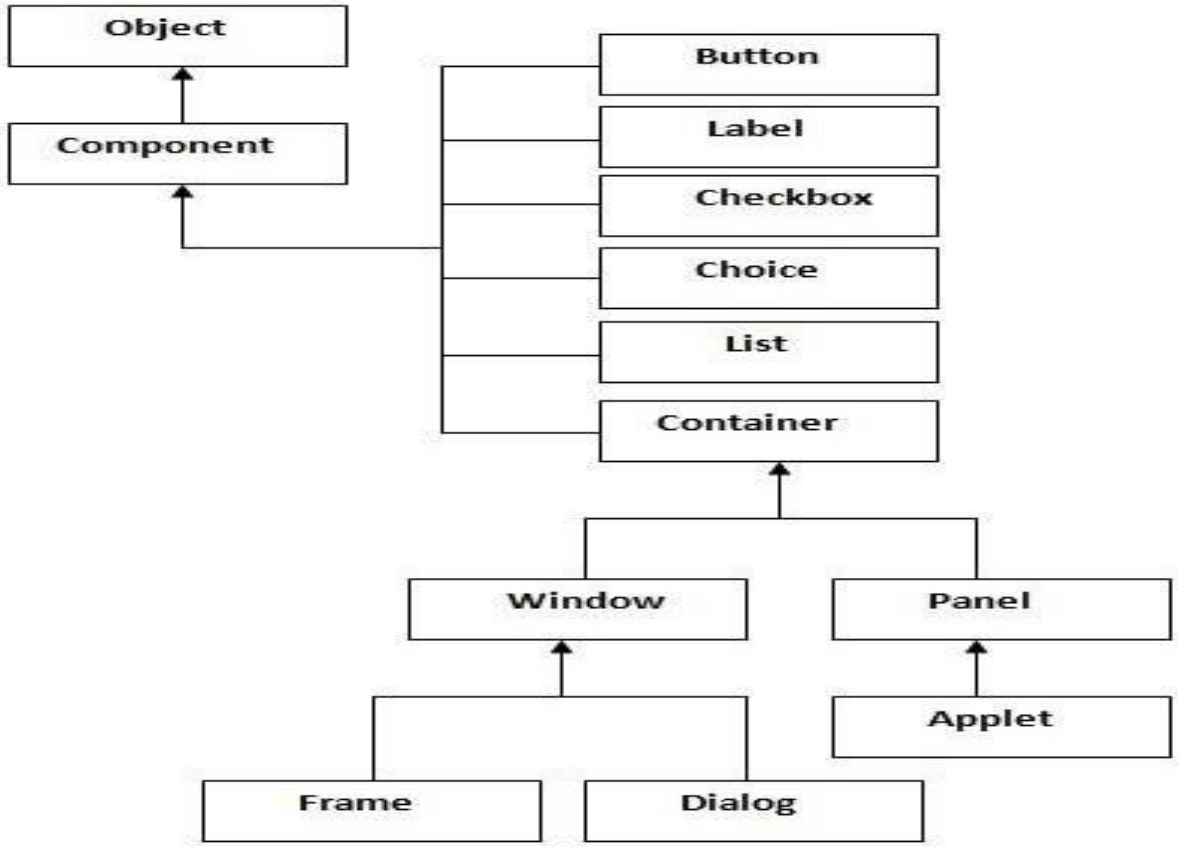
Button b= new Button("ok"); \\ Button( ), Button(String text)

```
f.add(l1);
f,add(l2);
f,add(t1);
f,add(t2);
f,add(b);
```



source listener

```
b.addActionListener(new ActionListener ( )
 {
  public void actionPerformed(ActionEvent    ae)
   {
    String temp= t1.getText();
   t1.setText(t2.getText());
   t2.settext(temp);
 }
}

     f.setSize(300,300);  //component- container-window-Frame
      //f.setVisible(true);
    }
```



Check Box

A check box is a control that is used to turn an option on or off. It consists of a small box that can either contain a check mark or not. There is a label associated with each check box that describes what option the box represents. You change the state of a check box by clicking on it. Check boxes can be used individually or as part of a group. Check boxes are objects of the Checkbox class.

Checkbox supports these constructors:

- Checkbox( ) throws HeadlessException
- Checkbox(String str) throws HeadlessException
- Checkbox(String str, boolean on) throws HeadlessException
- Checkbox(String str, boolean on, CheckboxGroup cbGroup) throws HeadlessException
- Checkbox(String str, CheckboxGroup cbGroup, boolean on) throws HeadlessException

The first form creates a check box whose label is initially blank. The state of the check box isunchecked.

The second form creates a check box whose label is specified by str. The state ofthe check box is unchecked. The third form allows you to set the initial state of the checkbox. If on is true, the check box is initially checked;otherwise, it is cleared.

The fourth and fifth forms create a check box whose label is specified by str and whose group is specified by cbGroup. If this check box is not part of a group, then cbGroup must be null. (Check box groups are described in the next section.) The value of on determines the initial state of thecheck box.

To retrieve the current state of a check box, call getState( ). To set its state, callsetState( ). You can obtain the current label associated with a check box by callinggetLabel( ). To set the label, call setLabel( ).

These methods are as follows:

boolean getState( )

void

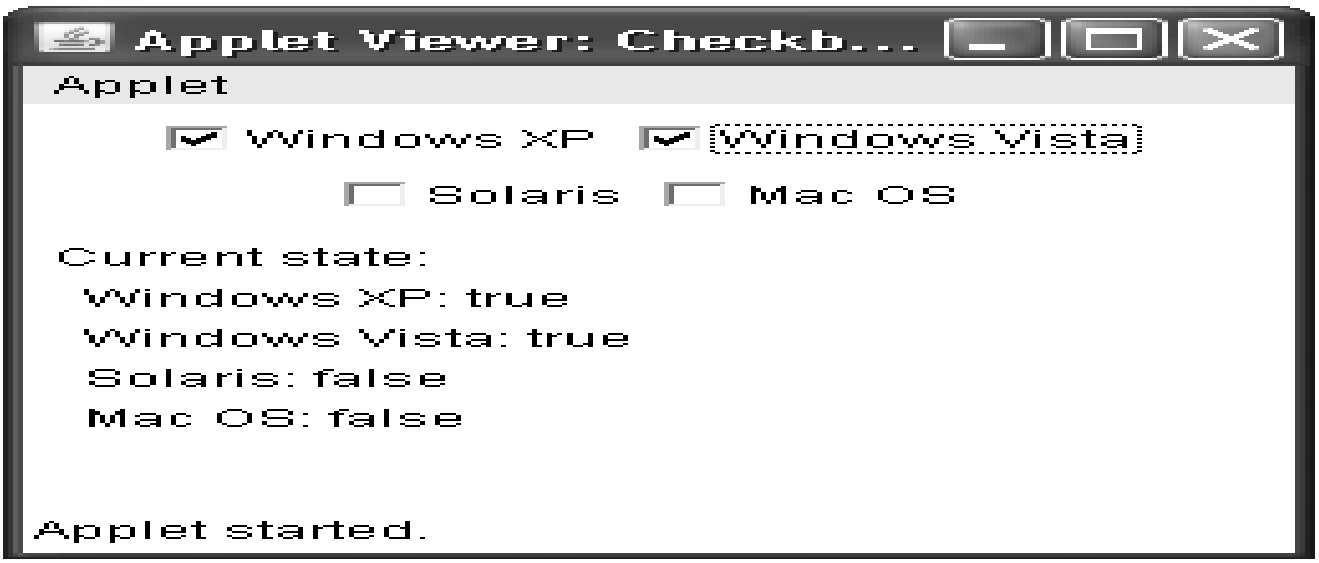
setState(boolean

on)String getLabel(

)

void setLabel(String str)Here, if on is true, the box is checked. If it is false, the box is cleared. The string passedin str becomes the new label associated with the invoking check box.

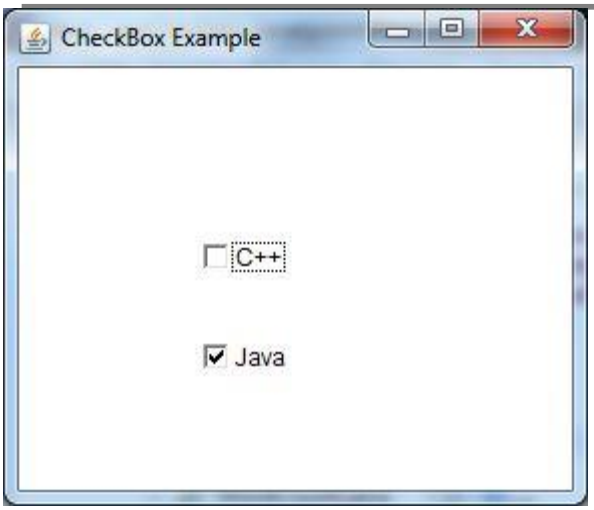
The below figure represents the appears of CheckBox



1. **public class Checkbox extends Component implements ItemSelectable, Accessible**

```
import java.awt.*;
public class CheckboxExample1
{
// constructor to initialize
    CheckboxExample1() {
// creating the frame with the title
      Frame f = new Frame("Checkbox Example");
// creating the checkboxes
       Checkbox checkbox1 = new Checkbox("C++");
       checkbox1.setBounds(100, 100,  50, 50);
       Checkbox checkbox2 = new Checkbox("Java", true);
// setting location of checkbox in frame
checkbox2.setBounds(100, 150,  50, 50);
// adding checkboxes to frame
       f.add(checkbox1);
       f.add(checkbox2);

// setting size, layout and visibility of frame
       f.setSize(400,400);
       f.setLayout(null);
       f.setVisible(true);
    }
// main method
public static void main (String args[])
{
   new CheckboxExample1();
}
}
```

Output:



CheckboxGroup

It is possible to create a set of mutually exclusive check boxes in which one and only one check box in the group can be checked at any one time. These check boxes are often called ***radio buttons.*** To create a set of mutually exclusive check boxes, you must first define the group to which they will belong and then specify that group when you constructer check boxes.

You can determine which check box in a group is currently selected by calling getSelectedCheckbox( ). You can set a check box by calling setSelectedCheckbox( ).

These methods are as follows:

Checkbox getSelectedCheckbox( )

void setSelectedCheckbox(Checkbox which)

Here, which is the check box that you want to be selected. The previously selected check box will be turned off.

The below figure represents the appears of CheckboxGroup

What are the differences between JToggle buttion and Radio buttion? [2marks]

Toggle Button:

Toggles should be used to represent an action, like turning something on or off, or starting or stopping an activity. It should be clear which state is on and which state is off. As the name suggest a button whose state can be toggled from on to off or vice-versa. For example a Switch in your home to turn a particular light on or off.

Toggle button has immediate effect on selection.

Radio Button:

Radio buttons should be used when the user can select one, and only one, option from a list of items. The button should be circular and become filled in when it is selected. Its name comes from the concept of buttons in Radio where for first station you press first button and for second station you press second button and so forth. So you can choose from multiple options. But at a time only one will be selected.

Radio Button has effect only after pressing Submit button

Choice

The Choice class is used to create a pop-up list of items from which the user may choose.Thus, a Choice control is a form of ***menu.*** When inactive, a Choice component takes up only enough space to show the currently selected item. When the user clicks on it, the whole list of choices pops up, and a new selection can be made. Each item in the list is a string that appears as a left-justified label in the order it is added to the Choice object. Choice only defines the default constructor, which creates an empty list.

- *To add a selection to the list, call add( ).* It has this general form:

void add(String name);

Here, name is the name of the item being added. Items are added to the list in the order inwhich calls to add( ) occur.

To determine which item is currently selected, you may call either getSelectedItem( )or getSelectedIndex( ). *These methods are shown here:*

String getSelectedItem( );

int getSelectedIndex( );

The getSelectedItem( ) method returns a string containing the name of the item.getSelectedIndex( ) returns the index of the item. The first item is at index 0. By default,the first item added to the list is selected.

To obtain the number of items in the list, call getItemCount( ). You can set the currently selected item using the select( ) method with either a zero-based integer index or a string that will match a name in the list. Given an index you can obtain the name associated with the item at that index by calling getItem( ),

These methods are shown here:

int getItemCount( );

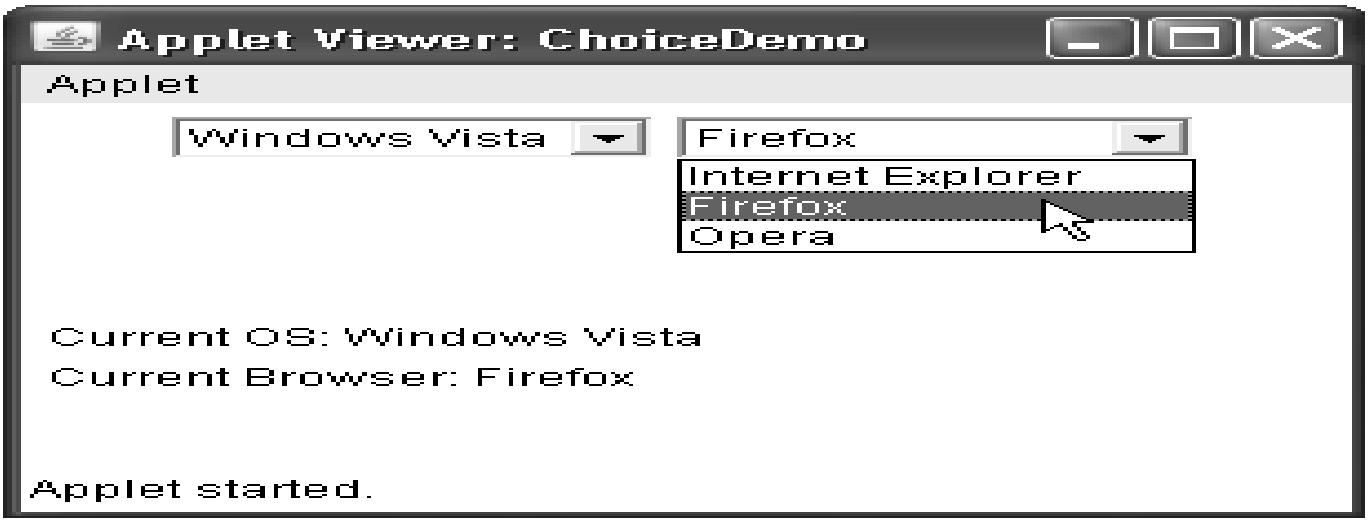
void select(int index);

void select(String name);

String getItem(int index);

Here, index specifies the index of the desired item

The below figure represents the appears of Choice



Lists

The List class provides a compact, multiple-choice, scrolling selection list. Unlike theChoice object, which shows only the single selected item in the menu, a List object can be constructed to show any number of choices in the visible window. It can also be created toallow multiple selections.

List constructors:

- *List( ) throws HeadlessException*
- *List(int numRows) throws HeadlessException*
- *List(int numRows, boolean multipleSelect) throws HeadlessException*

The first version creates a List control that allows only one item to be selected at any onetime.

In the second form, the value of numRows specifies the number of entries in the list that will always be visible (others can be scrolled into view as needed).

In the third form, if multipleSelect is true, then the user may select two or more items at a time. If it is false, then only one item may be selected.

To add a selection to the list, call add( ). It has the following two forms:

void add(String name);

void add(String name, int index);

Here, name is the name of the item added to the list. The first form adds items to the end of the list. The second form adds the item at the index specified by index. Indexing begins at zero. You can specify –1 to add the item to the end of the list.

For lists that allow only single selection, you can determine which item is currently selected by calling either getSelectedItem( ) or getSelectedIndex( ).

These methods are shown here:

String getSelectedItem( )

int getSelectedIndex( )

For lists that allow multiple selection, you must use either getSelectedItems( ) or getSelectedIndexes( ), shown here, to determine the current selections:

String[ ] getSelectedItems( )

int[ ] getSelectedIndexes( )

getSelectedItems( ) returns an array containing the names of the currently selected items.

getSelectedIndexes( ) returns an array containing the indexes of the currently selected items.

To obtain the number of items in the list, call getItemCount( ). You can set the currently

selected item by using the select( ) method with a zero-based integer index. Given an index, you can obtain the name associated with the item at that index bycalling getItem( )

These methods

are shown here:

int getItemCount( );

void select(int index);

String getItem(int index);

Here, index specifies the index of the desired item

The below figure represents the appears of List

TextField

The TextField class implements a single-line text-entry area, usually called an edit control.Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys, and mouse selections. TextField is a subclass of TextComponent.

TextField constructors:

- TextField( ) throws HeadlessException
- TextField(int numChars) throws HeadlessException
- TextField(String str) throws HeadlessException
- TextField(String str, int numChars) throws HeadlessException

The first version creates a default text field.

The second form creates a text field that is numChars characters wide.

The third form initializes the text field with the string contained in str.

The fourth form initializes a text field and sets its width.

TextField provides several methods that allow you to utilize a text field.

These methods are as follows:

String getText( ) ------To obtain the string currently contained in the text field,

void setText(String str) ------To set the text Here, str is the new string.

String getSelectedText( ) ------ program can obtain the currently selected text

void select(int startIndex, int endIndex)------ The user can select a portion of the text in a text field. Also, you can select a portion of text under program control. The select( ) method selects the charactersbeginning at startIndex and ending at endIndex–1.

boolean isEditable( ) ------returns true if the text may be changed and false if not

void setEditable(boolean canEdit) ----- = if canEdit is true, the text may be changed. If it is false, the text cannot be altered.

The below figure represents the appears of Textfield

TextArea

Sometimes a single line of text input is not enough for a given task. To handle these situations,the AWT includes a simple multiline editor called TextArea.

Constructorsn for TextArea:

- TextArea( ) throws HeadlessException
- TextArea(int numLines, int numChars) throws HeadlessException
- TextArea(String str) throws HeadlessException
- TextArea(String str, int numLines, int numChars) throws HeadlessException
- TextArea(String str, int numLines, int numChars, int sBars) throws HeadlessException

Here, numLines specifies the height, in lines, of the text area, and numChars specifies its width,

in characters. Initial text can be specified by str. In the fifth form, you can specify the scroll bars that you want the control to have. sBars must be one of these values: SCROLLBARS_BOTH, SCROLLBARS_NONE, SCROLLBARS_HORIZONTAL_ONLY, SCROLLBARS_VERTICAL_ONLY

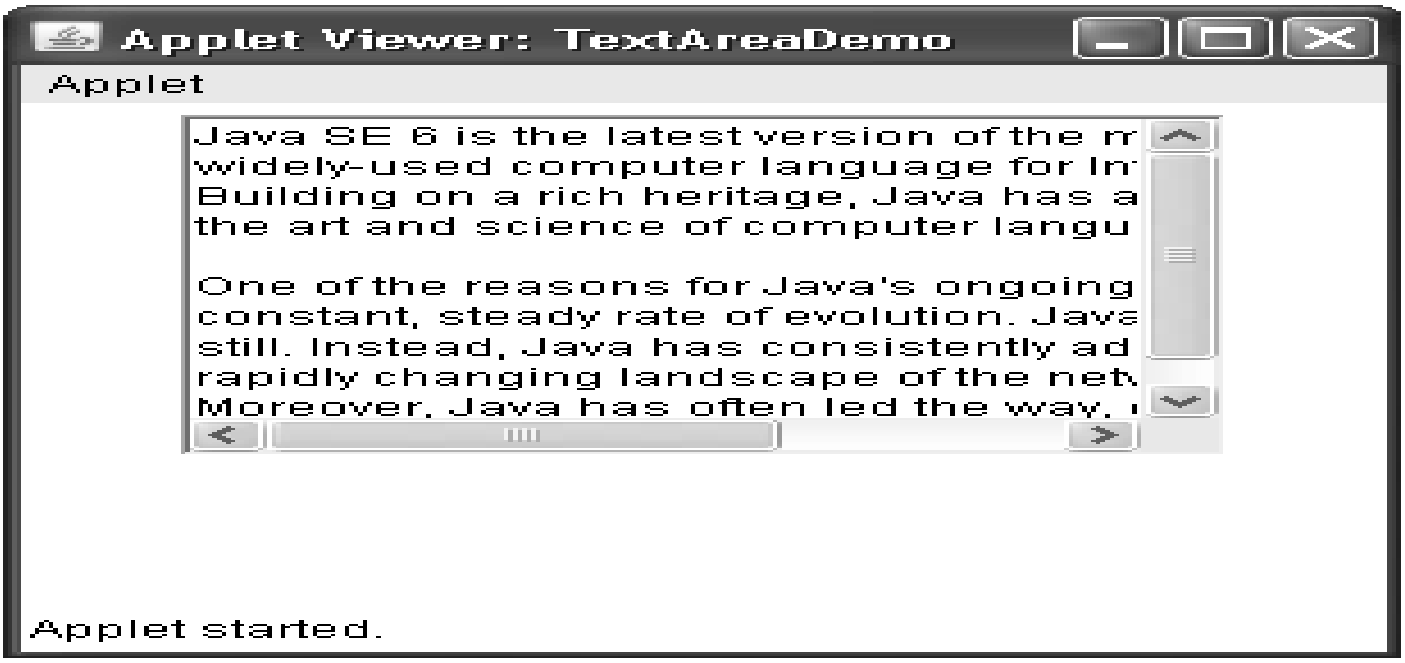
TextArea adds the following methods:

void append(String str); ------appends the string specified by str to the end of the current text

void insert(String str, int index); ------inserts the string passed in str at the specified index

void replaceRange(String str, int startIndex, int endIndex);------ It replaces the characters from startIndex to endIndex–1, with the replacement text passed in str

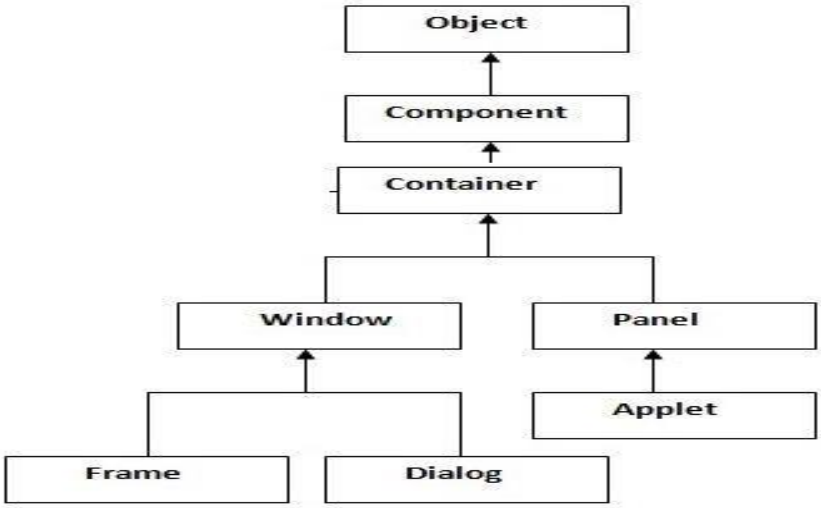
The below figure represents the appears of TextArea.



.5.1.5 Container

Abstract Windowing Toolkit (AWT): Abstract Windowing Toolkit (AWT) is used for GUI programming in java.

AWT Container Hierarchy:



Container:

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container.

Window:

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

Panel:

The Panel is the container that doesn't contain title bar and MenuBars. It can have other components like button, textfield etc.

Frame:

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

There are two ways to create a frame:

1. By extending Frame class (inheritance)

2. By creating the object of Frame class (association)

Example program to create a frame by extending Frame class (inheritance)

```
import java.awt.*;
class First extends Frame
{
First()
{
Button b=new Button("click me");
b.setBounds(30,100,80,30);/*setting button position public void setBounds(int xaxis, int yaxis, int width, int height); have been used in the above example that sets the position of the button.*/
 add(b);//adding button into frame
```

```
setSize(300,300);//frame size 300 width and 300 height
setLayout(null);//no layout now bydefault BorderLayout
setVisible(true);//now frame willbe visible, bydefault not visible
}
public static void main(String args[])
{
First f=new First();
}
}
```

2.Example program to create a frame by creating the object of Frame class

```
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();
 }
}
```

5.1.6 Layout Manager [2 marks-usage]

A container has a so-called *layout manager* to arrange its components. The layout managers provide a level of abstraction to map your user interface on all windowing systems, so that the layout can be *platform-independent*.

AWT provides the following layout managers (in package java.awt):

- Flow Layout
- Border Layout
- Grid Layout
- Card Layout
- Grid Bag Layout

Container's setLayout() method

A container has a setLayout() method to set its layout manager:

// java.awt.Container

public void setLayout(LayoutManager mgr)

To set up the layout of a Container (such as Frame, JFrame, Panel, or JPanel), you have to:

1. Construct an instance of the chosen layout object, via new and constructor, e.g., new FlowLayout())
2. Invoke the setLayout() method of the Container, with the layout object created as the argument;
3. Place the GUI components into the Container using the add() method in the correct order; or into the correct zones.

For example,

// Allocate a Panel (container)

Panel pnl = new Panel();

// Allocate a new Layout object. The Panel container sets to this layout.

pnl.setLayout(new FlowLayout());

// The Panel container adds components in the proper order.

pnl.add(new JLabel("One"));

```
pnl.add(new JLabel("Two"));
pnl.add(new JLabel("Three"));
.....
```

Container's getLayout() method

You can get the current layout via Container's getLayout() method.

```
Panel pnl = new Panel();
System.out.println(pnl.getLayout());
    // java.awt.FlowLayout[hgap=5,vgap=5,align=center]
```

Panel's Initial Layout

Panel (and Swing's JPanel) provides a constructor to set its initial layout manager. It is because a primary function of Panel is to layout a group of component in a particular layout.

public void Panel(LayoutManager *layout*)

// Construct a Panel in the given layout

Note: By default, Panel (and JPanel) has FlowLayout

```
// For example, create a Panel in BorderLayout
Panel pnl = new Panel(new BorderLayout());
```

FlowLayout:

In the java.awt.FlowLayout, components are arranged from left-to-right inside the container in the order that they are added (via method aContainer.add(aComponent)). When one row is filled, a new row will be started. The actual appearance depends on the width of the display window.

Constructors

public **FlowLayout**();

public **FlowLayout**(int *alignment*);

public **FlowLayout**(int *alignment*, int *hgap*, int *vgap*);

Note: *alignment* :

FlowLayout.LEFT (or LEADING), FlowLayout.RIGHT (or TRAILING), or FlowLayout.CENTER

hgap, vgap: horizontal/vertical gap between the components

Default value:

By default: *hgap* = 5, *vgap* = 5, *alignment* = FlowLayout.CENTER

Example

```
import java.awt.*;
import java.awt.event.*;
 // An AWT GUI program inherits the top-level container java.awt.Frame
public class AWTFlowLayoutDemo extends Frame
{
   private Button btn1, btn2, btn3, btn4, btn5, btn6;
   // Constructor to setup GUI components and event handlers
   public AWTFlowLayoutDemo ()
       {
     setLayout(new FlowLayout());
       // "super" Frame sets layout to FlowLayout, which arranges the components
```

```
    // from left-to-right, and flow from top-to-bottom.
   btn1 = new Button("Button 1");
   add(btn1);
   btn2 = new Button("This is Button 2");
   add(btn2);
   btn3 = new Button("3");
   add(btn3);
   btn4 = new Button("Another Button 4");
   add(btn4);
   btn5 = new Button("Button 5");
   add(btn5);
   btn6 = new Button("One More Button 6");
   add(btn6);
   setTitle("FlowLayout Demo"); // "super" Frame sets title
   setSize(280, 150);          // "super" Frame sets initial size
   setVisible(true);           // "super" Frame shows
}
// The entry main() method
public static void main(String[] args)
            {
   new AWTFlowLayoutDemo();  // Let the constructor do the job
             }
  }
```

GridLayout

In java.awt.GridLayout, components are arranged in a grid (matrix) of rows and columns inside the Container. Components are added in a left-to-right, top-to-bottom manner in the order they are added (via method *aContainer*.add(*aComponent*)).

Constructors

- public GridLayout(int *rows*, int *columns*);
- public GridLayout(int *rows*, int *columns*, int *hgap*, int *vgap*);

  // By default: rows = 1, cols = 0, hgap = 0, vgap = 0

Example

```
import java.awt.*;
import java.awt.event.*;
// An AWT GUI program inherits the top-level container java.awt.Frame
public class AWTGridLayoutDemo extends Frame
 {
   private Button btn1, btn2, btn3, btn4, btn5, btn6;
   // Constructor to setup GUI components and event handlers
   public AWTGridLayoutDemo () {
     setLayout(new GridLayout(3, 2, 3, 3));
        // "super" Frame sets layout to 3x2 GridLayout, horizontal and vertical gaps of 3 pixels
      // The components are added from left-to-right, top-to-bottom
```

```
    btn1 = new Button("Button 1");
    add(btn1);
    btn2 = new Button("This is Button 2");
    add(btn2);
    btn3 = new Button("3");
    add(btn3);
    btn4 = new Button("Another Button 4");
    add(btn4);
    btn5 = new Button("Button 5");
    add(btn5);
    btn6 = new Button("One More Button 6");
    add(btn6);
    setTitle("GridLayout Demo"); // "super" Frame sets title
    setSize(280, 150);          // "super" Frame sets initial size
    setVisible(true);           // "super" Frame shows
  }
  // The entry main() method
  public static void main(String[] args)
        {
    new AWTGridLayoutDemo();  // Let the constructor do the job
         }
}
```

BorderLayout

In java.awt.BorderLayout, the container is divided into 5 zones: EAST, WEST, SOUTH, NORTH, and CENTER. Components are added using method *aContainer*.add(*aComponent*, *zone*), where *zone* is either BorderLayout.NORTH (or PAGE_START), BorderLayout.SOUTH (or PAGE_END), BorderLayout.WEST (or LINE_START), BorderLayout.EAST (or LINE_END), or BorderLayout.CENTER.

You need not place components to all the 5 zones. The NORTH and SOUTH components may be stretched horizontally; the EAST and WEST components may be stretched vertically; the CENTER component may stretch both horizontally and vertically to fill any space left over.

Constructors

public **BorderLayout**();

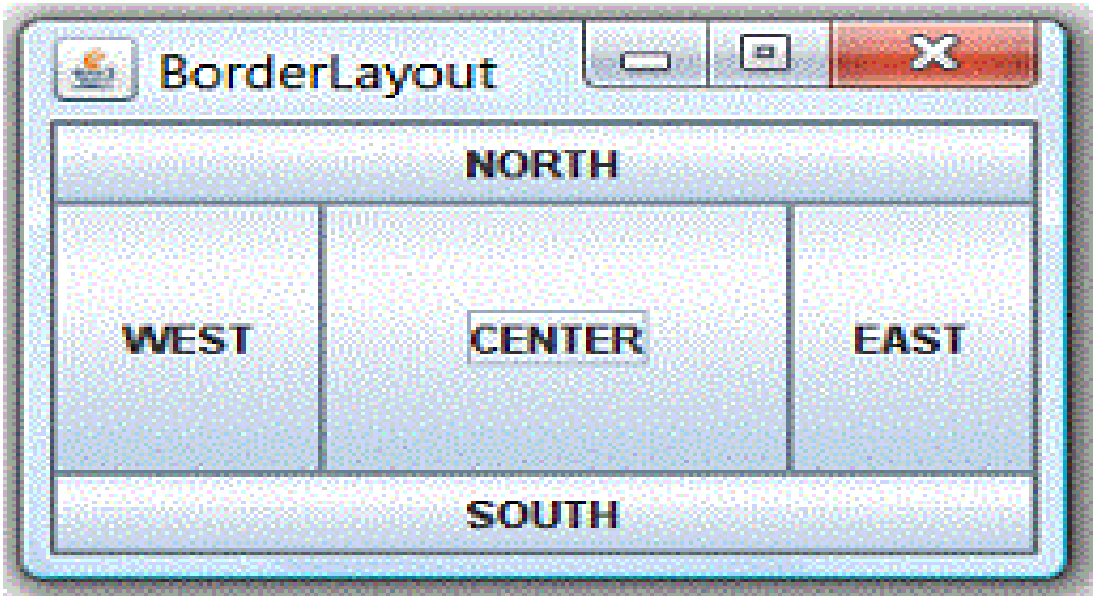
public **BorderLayout**(int hgap, int vgap);

// By default hgap = 0, vgap = 0

Example

```
import java.awt.*;

import java.awt.event.*;

// An AWT GUI program inherits the top-level container java.awt.Frame

public class AWTBorderLayoutDemo extends Frame

{

  private Button btnNorth, btnSouth, btnCenter, btnEast, btnWest;

  // Constructor to setup GUI components and event handlers

  public AWTBorderLayoutDemo ( )

    {

    setLayout(new BorderLayout(3, 3));

      // "super" Frame sets layout to BorderLayout, horizontal and vertical gaps of 3 pixels

    // The components are added to the specified zone

    btnNorth = new Button("NORTH");

    add(btnNorth, BorderLayout.NORTH);
```

```
      btnSouth = new Button("SOUTH");
      add(btnSouth, BorderLayout.SOUTH);
      btnCenter = new Button("CENTER");
      add(btnCenter, BorderLayout.CENTER);
      btnEast = new Button("EAST");
      add(btnEast, BorderLayout.EAST);
      btnWest = new Button("WEST");
      add(btnWest, BorderLayout.WEST);
      setTitle("BorderLayout Demo"); // "super" Frame sets title
      setSize(280, 150);            // "super" Frame sets initial size
      setVisible(true);             // "super" Frame shows
   }

   // The entry main() method
   public static void main(String[] args) {
      new AWTBorderLayoutDemo();  // Let the constructor do the job
   }
}
```



Question: differentiate between grid layout and border layout managers.[2 marks]

BorderLayout *- Lays out components in BorderLayout.NORTH, EAST, SOUTH, WEST, and CENTER sections.*		
bord =	**new BorderLayout**();	Creates BorderLayout. Widgets added with constraint to tell where.
bord =	**new BorderLayout**(*h*, *v*);	Creates BorderLayout with horizonal and vertical gaps sizes in pixels.
	p.**add**(*widget*, *pos*);	Adds *widget* to one of the 5 border layout regions, *pos* (see list above).
GridLayout *- Lays out components in equal sized rectangular grid, added r-t-l, top-to-bottom.*		
grid =	**new GridLayout**(*r*, *c*);	Creates GridLayout with specified rows and columns.
grid =	**new GridLayout**(*r*,*c*,*h*,*v*);	As above but also specifies horizontal and vertical space between cells.
	p.**add**(*widget*);	Adds *widget* to the next left-to-right, top-to-bottom cell.

GridLayout

- Arranges components into rows and columns
- A GridLayout puts all the components in a rectangular grid and is divided into equal-sized rectangles and each component is placed inside a rectangle
- A **GridLayout** is constructed with parameters
- When a component is added, it is placed in the next position in the grid, which is filled row by row, from the first to last column

BorderLayout

- Arranges components into five areas: North, South, East, West, and Center
- The class BorderLayout arranges the components to fit in the five regions: east, west, north, south and center. Each region is can contain only one component and each component in each region is identified by the corresponding constant NORTH, SOUTH, EAST, WEST, and CENTER.
- A **BorderLayout** can be constructed with no parameters
- We can add a single component at each of the four compass directions (specified by the Strings "North", "South", "East", or "West", as well as at the "Center". Of course, the component we add can be a container, which contains multiple components (managed by another layout manager).

Java CardLayout

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors of CardLayout class

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **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**.**

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

import javax.swing.*;

public class CardLayoutExample extends JFrame implements ActionListener{
CardLayout card;
JButton b1,b2,b3;
Container c;
   CardLayoutExample(){

       c=getContentPane();
       card=new CardLayout(40,30);
//create CardLayout object with 40 hor space and 30 ver space
       c.setLayout(card);

       b1=new JButton("Apple");
       b2=new JButton("Boy");
       b3=new JButton("Cat");
       b1.addActionListener(this);
       b2.addActionListener(this);
       b3.addActionListener(this);
```

```
        c.add("a",b1);c.add("b",b2);c.add("c",b3);

    }
    public void actionPerformed(ActionEvent e) {
    card.next(c);
    }

    public static void main(String[] args) {
        CardLayoutExample cl=new CardLayoutExample();
        cl.setSize(400,400);
        cl.setVisible(true);
        cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

GridBagLayout

The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.The components may not be of same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area. Each component associates an instance of GridBagConstraints. With the help of constraints object we arrange component's display area on the grid. The GridBagLayout manages each component's minimum and preferred sizes in order to determine component's size.

Useful Methods

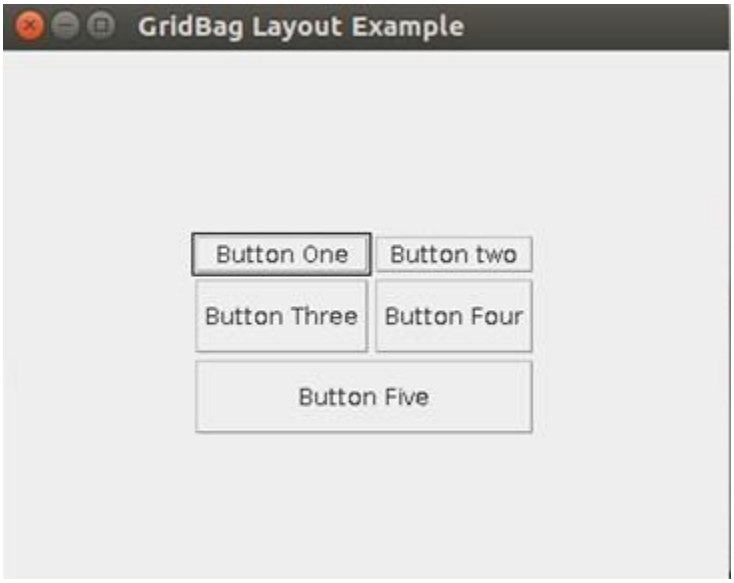
Modifier and Type	Method	Description
void	addLayoutComponent(Component comp, Object constraints)	It adds specified component to the layout, using the specified constraints object.
void	addLayoutComponent(String name, Component comp)	It has no effect, since this layout manager does not use a per-component string.
protected void	adjustForGravity(GridBagConstraints constraints, Rectangle r)	It adjusts the x, y, width, and height fields to the correct values depending on the constraint geometry and pads.

protected void	AdjustForGravity(GridBagConstraints constraints, Rectangle r)	This method is for backwards compatibility only
protected void	arrangeGrid(Container parent)	Lays out the grid.
protected void	ArrangeGrid(Container parent)	This method is obsolete and supplied for backwards compatibility
GridBagConstraints	getConstraints(Component comp)	It is for getting the constraints for the specified component.
float	getLayoutAlignmentX(Container parent)	It returns the alignment along the x axis.

Example program to demonstrate Java GridBagLayout

```
import java.awt.Button;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import javax.swing.*;
public class GridBagLayoutExample extends JFrame{
   public static void main(String[] args) {
        GridBagLayoutExample a = new GridBagLayoutExample();
      }
      public GridBagLayoutExample() {
   GridBagLayoutgrid = new GridBagLayout();
        GridBagConstraints gbc = new GridBagConstraints();
        setLayout(grid);
        setTitle("GridBag Layout Example");
        GridBagLayout layout = new GridBagLayout();
```

```
    this.setLayout(layout);
    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.gridx = 0;
    gbc.gridy = 0;
    this.add(new Button("Button One"), gbc);
    gbc.gridx = 1;
    gbc.gridy = 0;
    this.add(new Button("Button two"), gbc);
    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.ipady = 20;
    gbc.gridx = 0;
    gbc.gridy = 1;
    this.add(new Button("Button Three"), gbc);
    gbc.gridx = 1;
    gbc.gridy = 1;
    this.add(new Button("Button Four"), gbc);
    gbc.gridx = 0;
    gbc.gridy = 2;
    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.gridwidth = 2;
    this.add(new Button("Button Five"), gbc);
        setSize(300, 300);
        setPreferredSize(getSize());
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```



5.2 Event Handling

5.2.1 The Delegation event model

Java adopts the so-called "Event-Driven" (or "Event-Delegation") programming model for event-handling, similar to most of the visual programming languages, such as Visual Basic.

In event-driven programming, a piece of event-handling codes is executed (or called back by the graphics subsystem) when an event was fired in response to an user input (such as clicking a mouse button or hitting the ENTER key in a text field).

Call Back methods

In the above examples, the method actionPerformed() is known as a call back method. In other words, you never invoke actionPerformed() in your codes explicitly. The actionPerformed() is called back by the graphics subsystem under certain circumstances in response to certain user actions.

Source, Event and Listener Objects

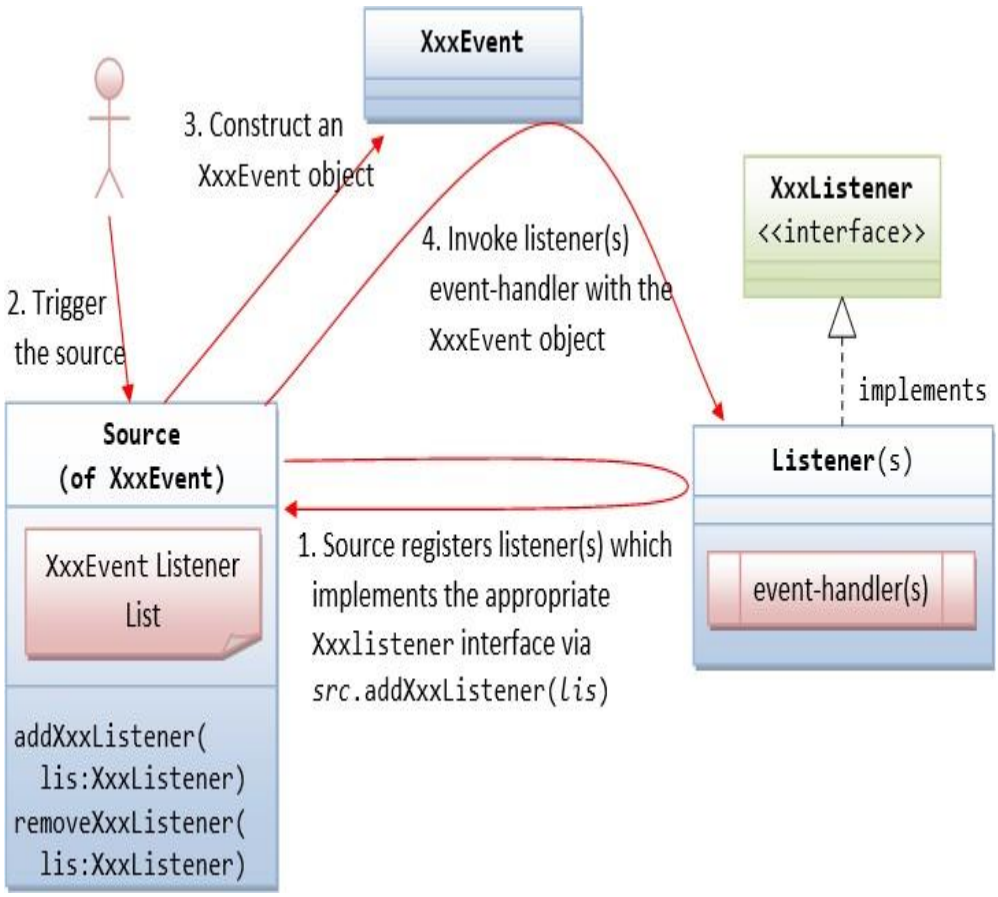
The AWT's event-handling classes are kept in package java.awt.event.

Three kinds of objects are involved in the event-handling: a source, listener(s) and an event object.

The source object (such as Button and Textfield) interacts with the user. Upon triggered, the source object creates an event object to capture the action (e.g., mouse-click x and y, texts entered, etc). This event object will be messaged to all the registered listener object(s), and an appropriate event-handler method of the listener(s) is called-back to provide the

response. In other words, triggering a source fires an event to all its listener(s), and invoke an appropriate event handler of the listener(s).

To express interest for a certain source's event, the listener(s) must be registered with the source. In other words, the listener(s) "subscribes" to a source's event, and the source "publishes" the event to all its subscribers upon activation. This is known as subscribe-publish or observable-observer design pattern.



The Delegation event model

The sequence of steps is illustrated above:

1. The source object registers its listener(s) for a certain type of event.

   (A source fires an event when triggered. For example, clicking a Button fires an ActionEvent, clicking a mouse button fires MouseEvent, typing a key fires KeyEvent, and etc.)

2. The source is triggered by a user.

3. The source create a XxxEvent object, which encapsulates the necessary information about the activation. For example, the (x, y) position of the mouse pointer, the text entered, etc.

4. Finally, for each of the XxxEvent listeners in the listener list, the source invokes the appropriate handler on the listener(s), which provides the programmed response

5.2.2 Events

Changing the state of an object is known as an **Event**. For example, click on button, dragging mouse etc.

The java.awt.event package provides many event classes and Listener interfaces for event handling.

It canbe generated as a consequence of a person interacting with the elements in a graphical user interface. Some of the activities that cause events to be generated are pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse. Many other user operations could also be cited as examples.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, or an operation is completed. You are free to define events that are appropriate for your application.

5.2.3 Event Sources

Asource 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. Asource 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 addTypeListener(TypeListener 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( ). The methodthat registers a mouse motion listener is called addMouseMotionListener( ). When an eventoccurs, all registered listeners are notified and receive a copy of the event object. ***This is known as multicasting the event.*** In all cases, notifications are sent only to listeners that register to receive them.

A source must also provide a method that allows a listener to unregister an interest in a specific type of event. The general form of such a method is this:

public void removeTypeListener(TypeListener el);

Here, Type is the name of the event, and el is a reference to the event listener. For example,to remove a keyboard listener, you would call removeKeyListener( ).The methods that add or remove listeners are provided by the source that generates events. For example, the Component class provides methods to add and remove keyboard and mouse event listeners.

5.2.4 Event Listerners

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.

The methods that receive and process events are defined in a set of interfaces found in java.awt.event.

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.

5.2.5 Event Classes

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.

EventObject contains two methods: getSource( ) and toString( ).

General form is shown here:

Object getSource( ); ------ returns the source of the event

String toString( ); --- returns the string equivalent of the event.

List of Event Classes and Listener Interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

5.2.6 Handling Mouse and Keyboard Events

Steps to perform EventHandling

Following steps are required to perform event handling :

1. Implement the Listener interface and overrides its methods
2. Register the component with the Listener

For registering the component with the Listener, many classes provide the registration methods.

For example:

- Button

  public void addActionListener(ActionListener a){}

- MenuItem

  public void addActionListener(ActionListener a){}

- TextField

  public void addActionListener(ActionListener a){}

public void addTextListener(TextListener a){}

- TextArea

  public void addTextListener(TextListener a){}

- Checkbox

  public void addItemListener(ItemListener a){}

- Choice

  public void addItemListener(ItemListener a){}

- List

  public void addActionListener(ActionListener a){}

  public void addItemListener(ItemListener a){}

Mouse Event:

- A MouseEvent is fired when you press, release, or click (press followed by release) a mouse-button (left or right button) at the source object; or position the mouse-pointer at (enter) and away (exit) from the source object.
- A MouseEvent listener must implement either MouseListener interface or MouseMotionListener interface or both dependending a action.
- The Java MouseListener is notified whenever you change the state of mouse
- The Java MouseMotionListener is notified whenever you move or drag mouse

Java MouseListener Interface

The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.

The signature of 5 methods found in MouseListener interface are given below:

1. public abstract void mouseClicked(MouseEvent e); // Called-back when the mouse-button has been clicked on the source.
2. public abstract void mouseEntered(MouseEvent e); //Called-back when the mouse-pointer has entered the source
3. public abstract void mouseExited(MouseEvent e); //Called-back when the mouse-pointer has exited the source

4. public abstract void mousePressed(MouseEvent e); // Called-back when a mouse-button has been pressed on the source

5. public abstract void mouseReleased(MouseEvent e); //Called-back when a mouse-button has been released on the source

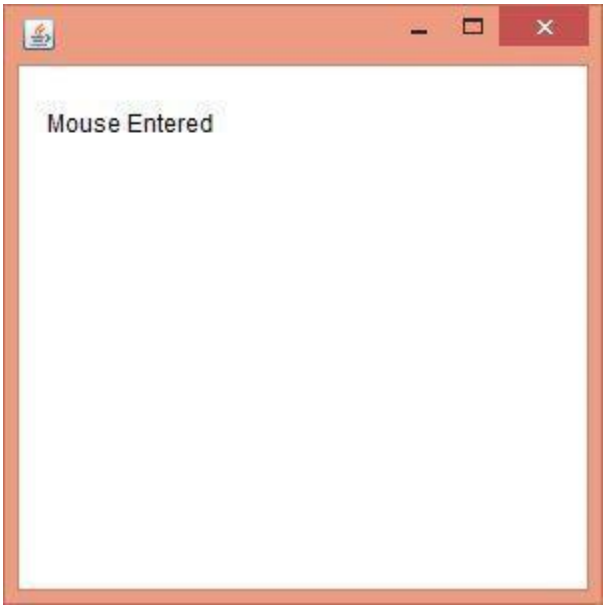
Note: A mouse-click invokes mousePressed(), mouseReleased() and mouseClicked().

Example on Java MouseListener:

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample extends Frame implements MouseListener{
   Label l;
   MouseListenerExample(){
      addMouseListener(this);

      l=new Label();
      l.setBounds(20,50,100,20);
      add(l);
      setSize(300,300);
      setLayout(null);
      setVisible(true);
   }
   public void mouseClicked(MouseEvent e) {
      l.setText("Mouse Clicked");
   }
   public void mouseEntered(MouseEvent e) {
      l.setText("Mouse Entered");
   }
   public void mouseExited(MouseEvent e) {
```

```
        l.setText("Mouse Exited");
    }
    public void mousePressed(MouseEvent e) {
        l.setText("Mouse Pressed");
    }
    public void mouseReleased(MouseEvent e) {
        l.setText("Mouse Released");
    }
public static void main(String[] args) {
    new MouseListenerExample();
}
}
```

Output:



Java MouseMotionListener Interface

The Java MouseMotionListener is notified whenever you move or drag mouse. It is notified against MouseEvent. The MouseMotionListener interface is found in java.awt.event package. It has two methods.

Methods of MouseMotionListener interface

The signature of 2 methods found in MouseMotionListener interface are given below:

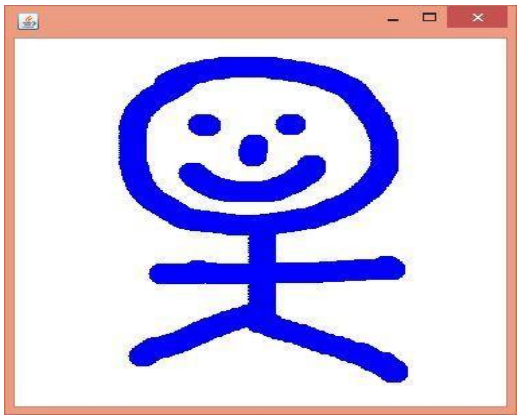
1. public abstract void mouseDragged(MouseEvent e);
2. public abstract void mouseMoved(MouseEvent e);

Example on Java MouseMotionListener ;

```
import java.awt.*;

import java.awt.event.*;

public class MouseMotionListenerExample extends Frame implements MouseMotionListener

{

   MouseMotionListenerExample(){

      addMouseMotionListener(this);


      setSize(300,300);

      setLayout(null);

      setVisible(true);

   }

public void mouseDragged(MouseEvent e) {

   Graphics g=getGraphics();

   g.setColor(Color.Green);

   g.fillOval(e.getX(),e.getY(),20,20);

}

public void mouseMoved(MouseEvent e) {}


public static void main(String[] args) {
```

```
    new MouseMotionListenerExample();
}
}
```

Output:



5.2.7 Adapter Classes

Java adapter classes provide the default implementation of listener interfaces. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it saves code.

The adapter classes are found in ***java.awt.event, java.awt.dnd*** and ***javax.swing.event*** packages. The Adapter classes with their corresponding listener interfaces are given below

java.awt.event Adapter classes:

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

java.awt.dnd Adapter classes

Adapter class	Listener interface
DragSourceAdapter	DragSourceListener
DragTargetAdapter	DragTargetListener

javax.swing.event Adapter classes:

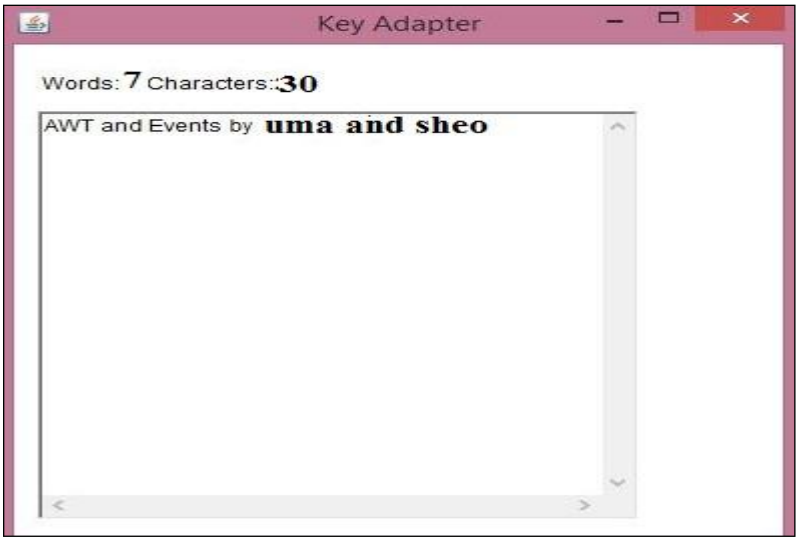
Adapter class	Listener interface
MouseInputAdapter	MouseInputListener
InternalFrameAdapter	InternalFrameListener

Example on Java KeyAdapter :

```
import java.awt.*;
import java.awt.event.*;
public class KeyAdapterExample extends KeyAdapter{
Label l;
TextArea area;
Frame f;
KeyAdapterExample(){
f=new Frame("Key Adapter");
l=new Label();
l.setBounds(20,50,200,20);
area=new TextArea();
area.setBounds(20,80,300, 300);
area.addKeyListener(this);
f.add(l);f.add(area);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
public void keyReleased(KeyEvent e) {
String text=area.getText();
String words[]=text.split("\\s");
l.setText("Words: "+words.length+" Characters:"+text.length());
}

  public static void main(String[] args) {
```

```
            new KeyAdapterExample();
            }
        }
```

Output:



5.2.8 Inner Classes

A nested class (or commonly called inner class) is a class defined inside another class As an illustration, two nested classes MyNestedClass1 and MyNestedClass2 are defined inside the definition of an outer class called MyOuterClass.

```
public class MyOuterClass {  // outer class defined here

  ......

  private class MyNestedClass1 { ..... }  // an nested class defined inside the outer class

  public static class MyNestedClass2 { ..... }  // an "static" nested class defined inside the outer class

  ......

}
```

Properties of nested class:

1. A nested class is a proper class. That is, it could contain constructors, member variables and member methods. You can create an instance of a nested class via the new operator and constructor.

2. A nested class is a member of the outer class, just like any member variables and methods defined inside a class.

3. Most importantly, a nested class can access the private members (variables/methods) of the enclosing outer class, as it is at the same level as these private members. This is the property that makes inner class useful.

4. A nested class can have private, public, protected, or the default access, just like any member variables and methods defined inside a class. A private inner class is only accessible by the enclosing outer class, and is not accessible by any other classes. [An top-level outer class cannot be declared private, as no one can use a private outer class.]

5. A nested class can also be declared static, final or abstract, just like any ordinary class.

6. A nested class is NOT a subclass of the outer class. That is, the nested class does not inherit the variables and methods of the outer class. It is an ordinary self-contained class. [Nonetheless, you could declare it as a subclass of the outer class, via keyword "extends OuterClassName", in the nested class's definition.]

The usages of nested class are:

1. To control visibilities (of the member variables and methods) between inner/outer class. The nested class, being defined inside an outer class, can access private members of the outer class.

2. To place a piece of class definition codes closer to where it is going to be used, to make the program clearer and easier to understand.

3. For namespace management.

Example program on inner classes

```
import java.awt.*;
import java.awt.event.*;
public class InnerClassDemo extends Frame {
    private TextField tfCount;
  private Button btnCount;
  private int count = 0;

  // Constructor to setup the GUI components and event handlers
  public InnerClassDemo () {
    setLayout(new FlowLayout());  // "super" Frame sets to FlowLayout
    add(new Label("Counter"));    // An anonymous instance of Label
    tfCount = new TextField("0", 10);
    tfCount.setEditable(false);    //  read-only
    add(tfCount);                 // "super" Frame adds tfCount
    btnCount = new Button("Count");
    add(btnCount);                // "super" Frame adds btnCount
    // Construct an anonymous instance of BtnCountListener (a named inner class).
    // btnCount adds this instance as a ActionListener.
    btnCount.addActionListener(new BtnCountListener());
    setTitle("AWT Counter");
    setSize(250, 100);
```

```
    setVisible(true);
  }
  // The entry main method
  public static void main(String[] args) {
    new InnerClassDem (); // Let the constructor do the job
  }
  /* BtnCountListener is a "named inner class" used as ActionListener. This inner class can access private
variables of the outer class.  */
  private class BtnCountListener implements ActionListener {

    public void actionPerformed(ActionEvent evt) //Override actionperformed method of ActionListener
{
      ++count;
      tfCount.setText(count + "");
    }
  }
}
```

Explaination of program:

- An inner class named BtnCountListener is used as the ActionListner.
- An anonymous instance of the BtnCountListener inner class is constructed. The btnCount source object adds this instance as a listener, as follows:

btnCount.addActionListener(new BtnCountListener());

- The inner class can access the private variable *tfCount* and *count* of the outer class.

5.2.9 Anonymous Inner Classes

Instead of using a named inner class (called BtnCountListner in the previous example), we shall use an inner class without a name, known as anonymous inner class as the ActionListener in this example

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

// An AWT GUI program inherits from the top-level container java.awt.Frame
public class AWTCounterAnonymousInnerClass extends Frame {
  // This class is NOT a ActionListener, hence, it does not implement ActionListener interface

  // The event-handler actionPerformed() needs to access these private variables
  private TextField tfCount;
  private Button btnCount;
  private int count = 0;
```

```
  // Constructor to setup the GUI components and event handlers
  public AWTCounterAnonymousInnerClass () {
    setLayout(new FlowLayout());  // "super" Frame sets to FlowLayout
    add(new Label("Counter"));    // An anonymous instance of Label
    tfCount = new TextField("0", 10);
    tfCount.setEditable(false);    //  read-only
    add(tfCount);                 // "super" Frame adds tfCount

    btnCount = new Button("Count");
    add(btnCount);                // "super" Frame adds btnCount

    // Construct an anonymous instance of an anonymous class.
    // btnCount adds this instance as a ActionListener.
    btnCount.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent evt) {
        ++count;
        tfCount.setText(count + "");
      }
    });

    setTitle("AWT Counter");
    setSize(250, 100);
    setVisible(true);
  }

  // The entry main method
  public static void main(String[] args) {
    new AWTCounterAnonymousInnerClass(); // Let the constructor do the job
  }
}
```

Applets

5.3 Swing Application and Applets

5.3.1 Applets and HTML

Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

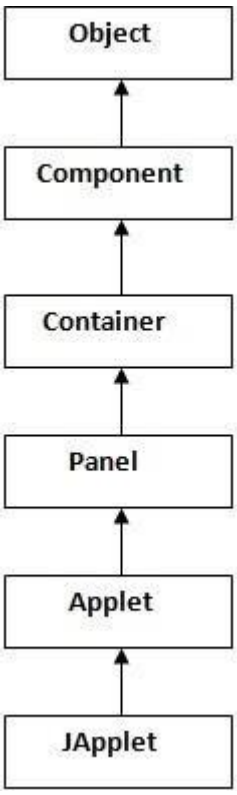
There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

- Plugin is required at client browser to execute applet.

Hierarchy of Applet



Lifecycle of Java Applet[2 marks]

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.

Lifecycle methods for Applet:

The java.applet.Applet class provides 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited.

It provides 4 life cycle methods of applet.

- public void init(): is used to initialized the Applet. It is invoked only once.
- public void start(): is invoked after the init() method or browser is maximized. It is used to start the Applet.
- public void stop(): is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
- public void destroy(): is used to destroy the Applet. It is invoked only once.

java.awt.Component class

The Component class provides 1 life cycle method of applet.

- **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

The HTML APPLET Tag

As mentioned earlier, Sun currently recommends that the APPLET tag be used to start an applet from both an HTML document and from an applet viewer.

The syntax for a fuller form of the APPLET tag is shown here. Bracketed items are optional.

< APPLET

[CODEBASE = codebaseURL]

CODE = appletFile

[ALT = alternateText]

[NAME = appletInstanceName]

WIDTH = pixels HEIGHT = pixels

[ALIGN = alignment]

[VSPACE = pixels] [HSPACE = pixels]

>

[< PARAM NAME = AttributeName VALUE = AttributeValue>]

[< PARAM NAME = AttributeName2 VALUE = AttributeValue>]

. . .

[HTML Displayed in the absence of Java]

</APPLET>

Let's take a look at each part now.

CODEBASE :

CODEBASE is an *optional* attribute that specifies the base URL of the applet code, which is the directory that will be searched for the applet's executable class file (specified by the CODE tag). The HTML document's URL directory is used as the CODEBASE if this attribute is not specified. The CODEBASE does not have to be on the host from which the HTML document was read.

CODE :

It is a required attribute that gives the name of the file containing your applet's compiled .class file. This file is relative to the code base URL of the applet, which is the directory that the HTML file was in or the directory indicated by CODEBASE if set.

ALT :

The ALT tag is an o*ptional* attribute used to specify a short text message that should be displayed if the browser recognizes the APPLET tag but can't currently run Java applets. This is distinct from the alternate HTML you provide for browsers that don't support applets.

NAME :NAME is an *optional* attribute used to specify a name for the applet instance.Applets must be named in order for other applets on the same page to find them by name and communicate with them. To obtain an applet by name, use getApplet( ), which is defined by the AppletContext interface.

WIDTH and HEIGHT :

WIDTH and HEIGHT are required attributes that give the size (in pixels) of the applet display area.

ALIGN:

ALIGN is an *optional* attribute that specifies the alignment of the applet. The possible values: LEFT, RIGHT,

TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE, and ABSBOTTOM.

VSPACE and HSPACE :

These attributes are optional. VSPACE specifies the space, in pixels,bove and below the applet. HSPACE specifies the space, in pixels, on each side of the applet.

PARAM NAME and VALUE :

The PARAM tag allows you to specify applet-specific arguments in an HTML page. Applets access their attributes with the getParameter( ) method.

Other valid APPLET attributes include ARCHIVE, which lets you specify one or more archive files, and OBJECT, which specifies a saved version of the applet.

Note:

APPLET tag should include only a CODE or an OBJECT attribute, but not both.

Question: Why do applet classes need to be declared as public? [2 marks]

Applet classes need to be declared as a public becomes the applet classes are access from HTML document that means from outside code. Otherwise, an applet will not be accessible from an outside code i.e an HTML

Running an Applet

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

1.Running an applet by html file :

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
//First.java

import java.applet.Applet;

import java.awt.Graphics;

public class First extends Applet{

public void paint(Graphics g){

g.drawString("welcome",150,150);

}
```

```
}
```

Note: class must be public because its object is created by Java Plugin software that resides on the browser.

myapplet.html

```
<html>
<body>
<applet code="First.class" width="300" height="300">
</applet>
</body>
</html>
```

2:Running Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer Filename.java. Now Html file is not required but it is for testing purpose only.

Example:

```
//First.java
import java.applet.Applet;
import  java.awt.Graphics;
public class First extends Applet
{
public void paint(Graphics g)
{
g.drawString("welcome to applet",150,150);
}
}
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java
c:\>appletviewer First.java
```

EventHandling in Applet

As we perform event handling in AWT or Swing, we can perform it in applet also. Let's see the simple example of event handling in applet that prints a message by click on the button**.**

Example of EventHandling in applet:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class EventApplet extends Applet implements ActionListener
{
Button b; TextField tf;
public void init()
        {
        tf=new TextField();
        tf.setBounds(30,40,150,20);
        b=new Button("Click");
        b.setBounds(80,150,60,50);
        add(b);
        add(tf);
        b.addActionListener(this);
        setLayout(null);
        }
public void actionPerformed(ActionEvent e)
        {
        tf.setText("Welcome");
         }
}
```

In the above example, we have created all the controls in init() method because it is invoked only once.

myapplet.html

```
<html>
<body>
<applet code="EventApplet.class" width="300" height="300">
</applet>
</body>
</html>
```

5.3.2 Security Issues

As you are likely aware, every time you download a “normal” program, you are taking a risk, because the code you are downloading might contain a virus, Trojan horse, or other harmful code. At the core of the problem is the fact that malicious code can cause its damage because it has gained unauthorized access to system resources. For example, a virus program might gather private information, such as credit card numbers, bank account balances, and passwords, by searching the contents of your computer’s local file system.

In order for Java to enable applets to be downloaded and executed on the client computer safely, it was

necessary to prevent an applet from launching such an attack.Java achieved this protection by confining an applet to the Java execution environmentand not allowing it access to other parts of the computer. The ability to download applets with confidence that no harm will be done and that no security will be breached is considered by many to be the single most innovative aspect of Java

5.3.3 Applets and Applications

Advantages of Applets:

1. Execution of applets is easy in a Web browser and does not require any installation or deployment procedure in realtime programming (where as servlets require).

2. Writing and displaying (just opening in a browser) graphics and animations is easier than applications.

3. In GUI development, constructor, size of frame, window closing code etc. are not required (but are required in applications).

Restrictions of Applets:

1. Applets are required separate compilation before opening in a browser.

2. In realtime environment, the bytecode of applet is to be downloaded from the server to the client machine.

3. Applets are treated as untrusted (as they were developed by unknown people and placed on unknown servers whose trustworthiness is not guaranteed) and for this reason they are not allowed, as a security measure, to access any system resources like file system etc. available on the client system.

4. Extra Code is required to communicate between applets using AppletContext

Differences between applications and applets-[3 marks]

FEATURE	APPLET	APPLICATION
main() method	Not Present	present
Nature	Requires some third party tool help like a browser to execute	Called as stand-alone application as application can be executed from command prompt
Restrictions	cannot access any thing on the system except browser's services	Can access any data or software available on the system
Security	Requires highest security for the system as they are untrusted	Does not require any security
Execution	Applet is portable and can be executed by any JAVA supported browser	Need JDK, JRE, JVM installed on client machine
Creation	Applets are created by extending the java.applet.Applet	Applications are created by writing public static void main(String[] s) method
Methods	Applet application has 5 methods which will be automatically invoked on occurance of specific event	Application has a single start point which is main method
Example	Example:	Example:

	import java.awt.*;
	import java.applet.*;
	
	public class Myclass extends Applet
	{
	public void init() { }
	public void start() { }
	public void stop() {}
	public void destroy() {}
	public void paint(Graphics g) {}
	}
		public class MyClass
		{
		public static void main(String args[]) {}
		}

5.3.4 Passing Parameters to Applets

We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named getParameter().

Syntax:

public String getParameter(String parameterName);

Example on Passing Parameters to Applets

```
import java.applet.Applet;
import java.awt.Graphics;
public class UseParam extends Applet
{
    public void paint(Graphics g)
```

```
{
String str=getParameter("msg");
g.drawString(str,50, 50);
}
}
```

myapplet.html

```
<html>
<body>
<applet code="UseParam.class" width="300" height="300">
<param name="msg" value="Welcome to applet">
</applet>
</body>
</html>
```

5.3.5 Creating a Swing Applet

We can even create applets based on the Swing package. In order to create such applets, we must extend JApplet class of the swing package. JApplet extends Applet class, hence all the features of Applet class are available in JApplet as well, including JApplet's own Swing based features. Swing applets provides an easier to use user interface than AWT applets.

Some important points :

- When we creat an AWT applet, we implement the paint(Graphics g) method to draw in it but when we create a Swing applet, we implement paintComponent(Graphics g) method to draw in it.
- For an applet class that extends Swing's JApplet class, the way elements like textfield, buttons, checksboxes etc are added to this applet is performed by using a default layout manager, i.e. BorderLayout.

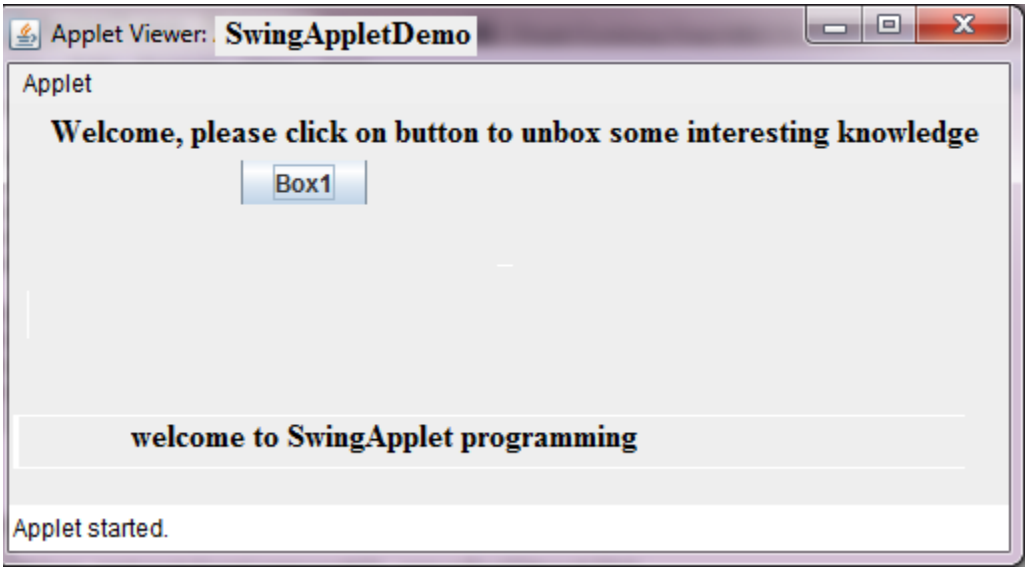
Swing applets use the same four lifecycle methods: init( ),start( ), stop( ), and destroy( ). Of course, you need override only those methods that are needed by your applet. Painting is accomplished differently in Swing than it is in the AWT,and a Swing applet will not normally override the paint( ) method.

Example on Creating a swing applet

In this example , first, we have created a swing applet by extending JApplet class and we have added a JPanel to it.Next, we have created a class B, which has extended JPanel class of Swing package and have also implemented ActionListener interace to listen to the button click event generated when buttons added to JPanel are clicked.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="SwingAppletDemo" width="500" height="200">
</applet>
*/
public class SwingAppletDemo extends JApplet
{
public void init()
{
add(new B());              //Adding a JPanel to this Swing applet
}
}


class B extends JPanel implements ActionListener
{
JLabel  jb;
JButton box1;
String str;

B( )
        {
        jb= new JLabel("Welcome, please click on button to unbox some interesting knowledge -");
        box1 = new JButton("Box1");
        str ="";
        setLayout(new FlowLayout());
        add(jb);
        add(box1);
        box1.addActionListener(this);
        }
public void actionPerformed(ActionEvent ae)
{
```

```
if(ae.getActionCommand().equals("box1"))
        {
str=" welcome to SwingApplet programming";
repaint();
        }
}

public void paintComponent(Graphics g)
{
super.paintComponent(g);
g.drawString(str, 2, 170);
}
}
```



5.3.6 Painting in Swing

Painting is accomplished differently in Swing than it is in the AWT,and a Swing applet will not normally override the paint( ) method.when we create a Swing applet, we implement paintComponent(Graphics g) method to draw in it.

Syntax:

protected void paintComponent(Graphics g)

The parameter g is the graphics context to which output is written.

To cause a component to be painted under program control, call repaint( ). It works in Swing just as it does for the AWT. The repaint( ) method is defined by Component. Calling it causes the system to call paint( ) as soon

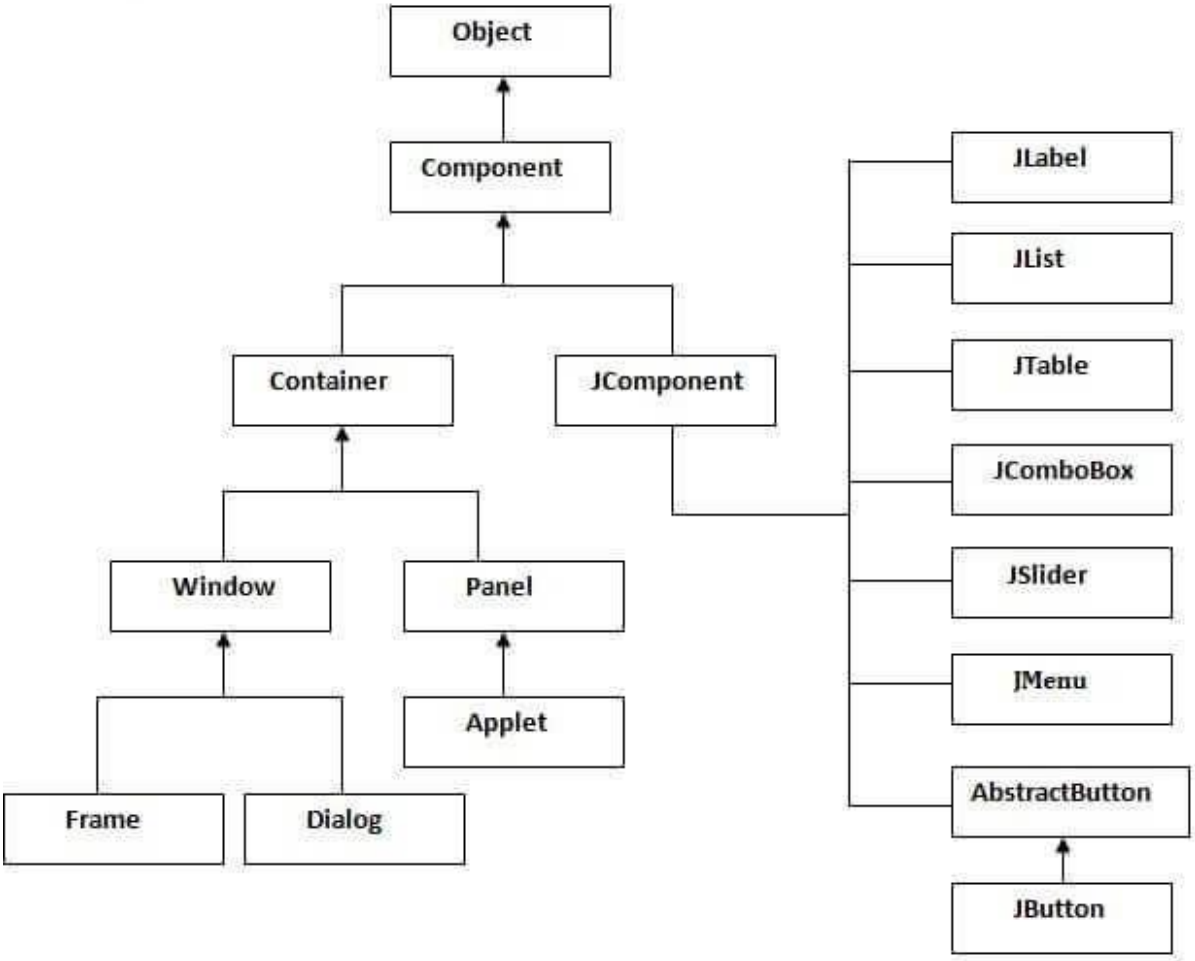
as it is possible to do so. In Swing the call to paint( ) results in a call to paintComponent( ). Inside the overridden paintComponent( ), you will draw the stored output

Note: For example refer to the program on Creating a Swing Applet

5.3.7 Exploring Swing Controls- JLabel , Image Icon and JText Field

Hierarchy for swing components. [3marks]

the hierarchy of java swing API is given below.



Hierarchy of swing components

Exploring Swing Controls

- **JLable , Image Icon and JTextField**

JLabels

A JLabel is a very simple component which contains a string.

The constructors are

- public JLabel() // creates label with no text
- public JLabel(String text) //create label with text

The methods available are

- public String getText() // return label text
- public void setText(String s) // sets the label text

Image Icon

The class ImageIcon is an implementation of the Icon interface that paints Icons from Images.

The constructors are

- ImageIcon() // Creates an uninitialized image icon.
- ImageIcon(Image image)// Creates an ImageIcon from an image object.
- ImageIcon(Image image, String description)// Creates an ImageIcon from the image.
- ImageIcon(String filename, String description)// Creates an ImageIcon from the specified file.
- ImageIcon(URL location)// Creates an ImageIcon from the specified URL.

The methods available are

- String getDescription() // Gets the description of the image
- Image getImage()//Returns this icon's Image.
- void paintIcon(Component c, Graphics g, int x, int y)// Paints the icon.
- void setDescription(String description)// Sets the description of the image.
- void setImage(Image image)// Sets the image displayed by this icon.
- String toString()//Returns a string representation of this image.

JTextField

A JTextField is an area that the user can type one line of text into. It is a good way of getting text input from the user.

The constructors are :

- public JTextField () // creates text field with no text
- public JTextField (int columns) // create text field with appropriate # of columns
- public JTextField (String s) // create text field with s displayed
- public JTextField (String s, int columns) // create text field with s displayed & approp. width

Methods include:

- public void setEditable(boolean s) // if false the TextField is not user editable
- public String getText() // return label text
- public void setText(String s) // sets the label text

.5.3.8 Swing Buttons

Java Swing 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.
Unlike AWT, Java Swing provides platform-independent and lightweight components.
The javax.swing package provides classes for java swing API such

JButton
JToggle Button
JCheckBox
JRadio Button
JTabbed Pane
JScroll Pane
JList
JComboBox
Swing Menus
Dialogs

JButton

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

JButton class declaration

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

1. public class JButton extends AbstractButton implements Accessible

Commonly used Constructors

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

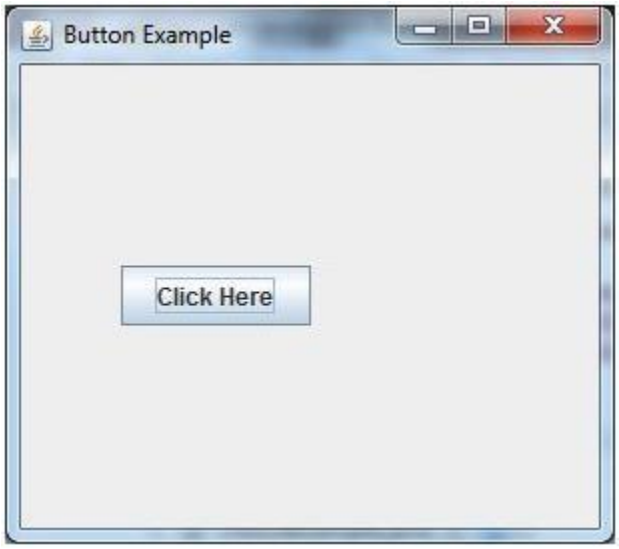
Commonly used Methods of AbstractButton class:

Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Java JButton 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:



Question:
subclasses of JButton class.[2 marks]

Subclasses:
1.BasicArrowButton,
2.MetalComboBoxButton

1.BasicArrowButton:

public class BasicArrowButton extends JButton implements SwingConstants MetalComboBoxButton

Constructors:
BasicArrowButton(int direction); --Creates a BasicArrowButton whose arrow is drawn in the specified direction.

BasicArrowButton(int direction, Color background, Color shadow, Color darkShadow, Color highlight)-- Creates a BasicArrowButton whose arrow is drawn in the specified direction and with the specified colors.

2.MetalComboBoxButton

Constructors :

MetalComboBoxButton(JComboBox cb, Icon i, boolean onlyIcon, CellRendererPane pane, JList list)

MetalComboBoxButton(JComboBox cb, Icon i, CellRendererPane pane, JList list)

JToggle Button

Auseful variation on the push button is called a toggle button. A toggle button looks just like a push button, but it acts differently because it has two states: pushed and released. That is, when you press a toggle button, it stays pressed rather than popping back up as a regular push button does. When you press the toggle button a second time, it releases (pops up). Therefore, each time a toggle button is pushed, it toggles between its two states

JToggleButton constructors:

JToggleButton(String str)

This creates a toggle button that contains the text passed in str. By default, the button is in the off position. Other constructors enable you to create toggle buttons that contain images,or images and text.

To handle item events, you must implement the ItemListener interface.Each time an item event is generated, it is passed to the itemStateChanged( )method defined by ItemListener. Inside itemStateChanged( ), the getItem( ) method canbe called on the ItemEvent object to obtain a reference to the JToggleButton instance thatgenerated the event. It is shown here:

Object getItem( )

A reference to the button is returned. You will need to cast this reference to JToggleButton.The easiest way to determine a toggle button's state is by calling the isSelected( ) method(inherited from AbstractButton) on the button that generated the event. It is shown here:

boolean isSelected( )

It returns true if the button is selected and false otherwise

Example program to demonstrate JToggleButton

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JToggleButtonDemo" width=200 height=80>
</applet>
*/
public class JToggleButtonDemo extends JApplet {
JLabel jlab;
JToggleButton jtbn;
public void init() {
try {
SwingUtilities.invokeAndWait(
new Runnable() {
public void run() {
makeGUI();
}
}
);
} catch (Exception exc) {
System.out.println("Can't create because of " + exc);
}
}
private void makeGUI() {
// Change to flow layout.
setLayout(new FlowLayout());
// Create a label.
jlab = new JLabel("Button is off.");
// Make a toggle button.
jtbn = new JToggleButton("On/Off");
// Add an item listener for the toggle button.
jtbn.addItemListener(new ItemListener() {
public void itemStateChanged(ItemEvent ie) {
if(jtbn.isSelected())
jlab.setText("Button is on.");
else
jlab.setText("Button is off.");
}
});
// Add the toggle button and label to the content pane.
add(jtbn);
```

```
add(jlab);
}
}
```

The output from the toggle button example is shown here:

JCheck Box

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits JToggleButton class.

JCheckBox class declaration

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

public class JCheckBox extends JToggleButton implements Accessible

Commonly used Constructors:

Constructor	Description
JJCheckBox()	Creates an initially unselected check box button with no text, no icon.
JChechBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(Action a)	Creates a check box where properties are taken from the Action supplied.

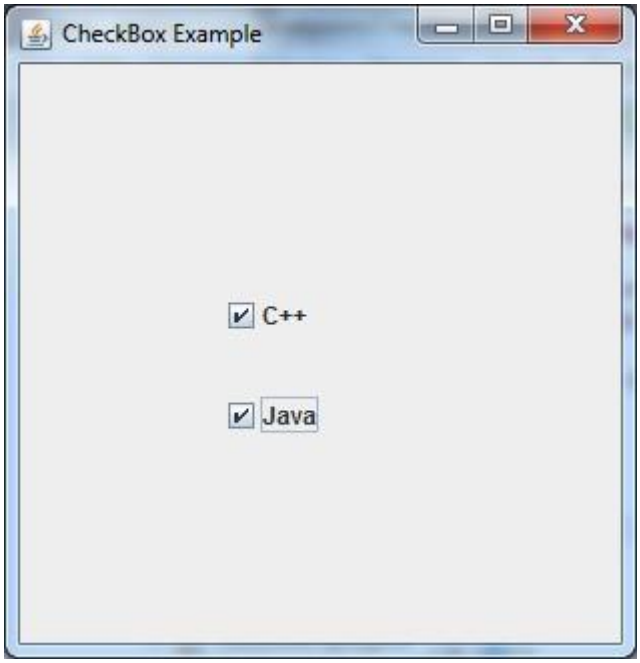
Commonly used Methods:

Methods	Description
AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this JCheckBox.
protected String paramString()	It returns a string representation of this JCheckBox.

Example program to demonstrate JCheckBox

```
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:



JRadio Button

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

It should be added in ButtonGroup to select one radio button only.

JRadioButton class declaration

public class JRadioButton extends JToggleButton implements Accessible

Commonly used Constructors:

Constructor	Description
JRadioButton()	Creates an unselected radio button with no text.
JRadioButton(String s)	Creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected)	Creates a radio button with the specified text and selected status.

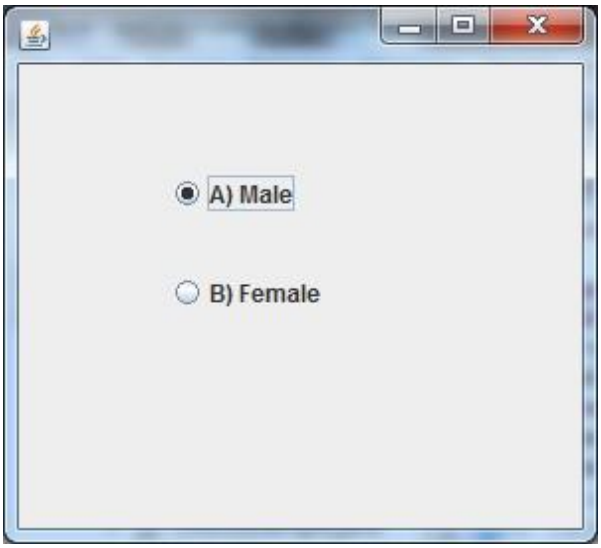
Commonly used Methods:

Methods	Description
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Example program to demonstrate JRadioButton

```
import javax.swing.*;
public class RadioButtonExample
{
JFrame 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:



JTabbed Pane

JTabbedPane encapsulates a tabbed pane. It manages a set of components by linking them with tabs. Selecting a tab causes the component associated with that tab to come to the forefront.

JTabbedPane defines three constructors. We will use its default constructor, whichcreates an empty control with the tabs positioned across the top of the pane. The other two constructors let you specify the location of the tabs, which can be along any of the foursides. JTabbedPane uses the SingleSelectionModel model.Tabs are added by calling addTab( ). Here is one of its forms:

void addTab(String name, Component comp)

Here, *name* is the name for the tab, and *comp* is the component that should be added to the tab. Often, the component added to a tab is a JPanel that contains a group of related components. This technique allows a tab to hold a set of components

The general procedure to use a tabbed pane is outlined here:

1. Create an instance of JTabbedPane.

2. Add each tab by calling addTab( ).

3. Add the tabbed pane to the content pane.

Example to illustrates a Tabbed Pane.

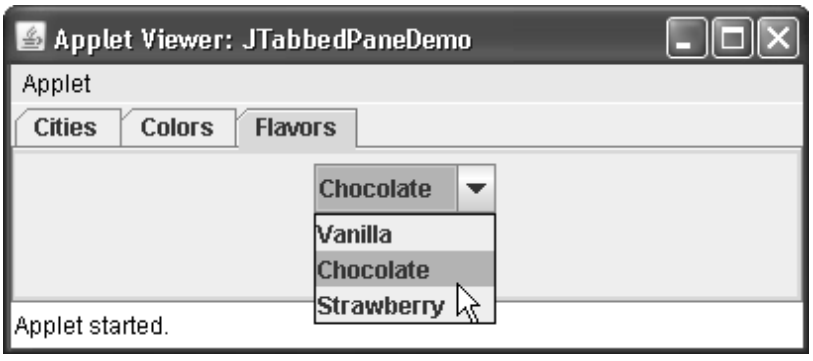
In this exampleThe first tab is titled “Cities” andcontains four buttons. Each button displays the name of a city. The second tab is titled“Colors” and contains three check boxes. Each check box displays the name of a color. The third tab is titled “Flavors” and contains one combo box. This enables the user to select oneof three flavors.

```
// Demonstrate JTabbedPane.
import javax.swing.*;
/*
<applet code="JTabbedPaneDemo" width=400 height=100>
</applet>
*/
public class JTabbedPaneDemo extends JApplet {
public void init() {
try {
SwingUtilities.invokeAndWait(
new Runnable() {
public void run() {
```

```
makeGUI();
}
}
);
} catch (Exception exc) {
System.out.println("Can't create because of " + exc);
}
}
private void makeGUI() {
JTabbedPane jtp = new JTabbedPane();
jtp.addTab("Cities", new CitiesPanel());
jtp.addTab("Colors", new ColorsPanel());
jtp.addTab("Flavors", new FlavorsPanel());
add(jtp);
}
}
// Make the panels that will be added to the tabbed pane.
class CitiesPanel extends JPanel {
public CitiesPanel() {
JButton b1 = new JButton("New York");
add(b1);
JButton b2 = new JButton("London");
add(b2);
JButton b3 = new JButton("Hong Kong");
add(b3);
JButton b4 = new JButton("Tokyo");
add(b4);
}
}
class ColorsPanel extends JPanel {
public ColorsPanel() {
JCheckBox cb1 = new JCheckBox("Red");
add(cb1);
JCheckBox cb2 = new JCheckBox("Green");
add(cb2);
JCheckBox cb3 = new JCheckBox("Blue");
add(cb3);
}
}
class FlavorsPanel extends JPanel {
public FlavorsPanel() {
JComboBox jcb = new JComboBox();
jcb.addItem("Vanilla");
```

```
jcb.addItem("Chocolate");
jcb.addItem("Strawberry");
add(jcb);
}
}
```

Output:



JScroll Pane

JScrollPane is a lightweight container that automatically handles the scrolling of another component. The component being scrolled can either be an individual component, such as a table, or a group of components contained within another lightweight container, such as a JPanel

JScrollPane defines several constructors. The one used in this chapter is shown here:

JScrollPane(Component comp)

The component to be scrolled is specified by comp. Scroll bars are automatically displayed when the content of the pane exceeds the dimensions of the viewport.

Here are the steps to follow to use a scroll pane:
1. Create the component to be scrolled.
2. Create an instance of JScrollPane, passing to it the object to scroll.
3. Add the scroll pane to the content pane.

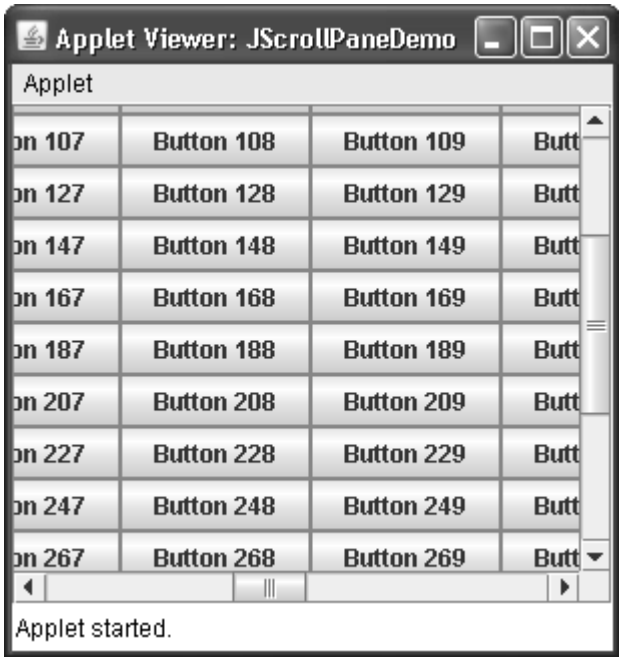
Example program to illustrates JScroll Pane

First, a JPanel object is created, and 400 buttons are added to it, arranged into 20 columns. This panel is then added to a scroll pane,and the scroll pane is added to the content pane. Because the panel is larger than the viewport, vertical and horizontal scroll bars appear automatically. You can use the scroll bars to scroll the buttons into view.

```
// Demonstrate JScrollPane.
import java.awt.*;
import javax.swing.*;
/*
<applet code="JScrollPaneDemo" width=300 height=250>
</applet>
*/
public class JScrollPaneDemo extends JApplet {
public void init() {
try {
SwingUtilities.invokeAndWait(
new Runnable() {
public void run() {
makeGUI();
```

```
}
}
);
} catch (Exception exc) {
System.out.println("Can't create because of " + exc);
}
}
private void makeGUI() {
// Add 400 buttons to a panel.
JPanel jp = new JPanel();
jp.setLayout(new GridLayout(20, 20));
int b = 0;
for(int i = 0; i < 20; i++) {
for(int j = 0; j < 20; j++) {
jp.add(new JButton("Button " + b));
++b;
}
}
// Create the scroll pane.
JScrollPane jsp = new JScrollPane(jp);
// Add the scroll pane to the content pane.
// Because the default border layout is used,
// the scroll pane will be added to the center.
add(jsp, BorderLayout.CENTER);
}
}
```

Output:



JList

The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

JList class declaration

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

public class JList extends JComponent implements Scrollable, Accessible

Commonly used Constructors:

Constructor	Description
JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.
JList(ListModel<ary> dataModel)	Creates a JList that displays elements from the specified, non-null, model.

Commonly used Methods:

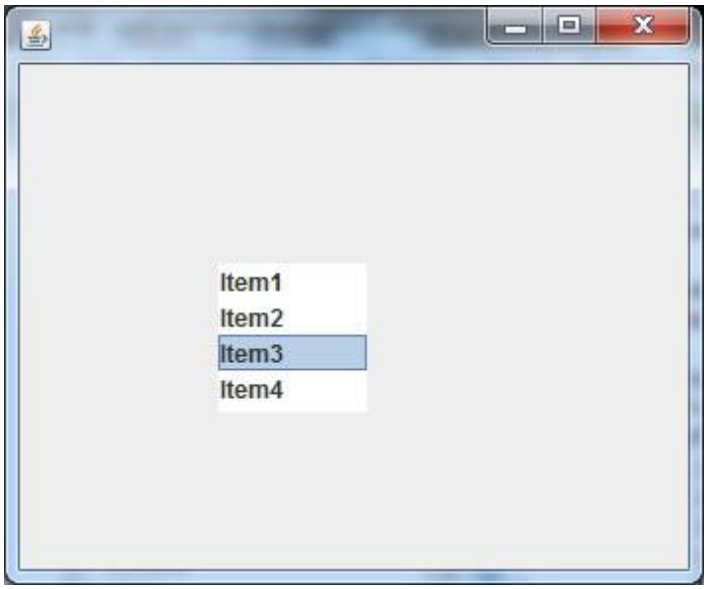
Methods	Description
Void addListSelectionListener(ListSelectionListener listener)	It is used to add a listener to the list, to be notified each time a change to the selection occurs.
int getSelectedIndex()	It is used to return the smallest selected cell index.
ListModel getModel()	It is used to return the data model that holds a list of items displayed by the JList component.
void setListData(Object[] listData)	It is used to create a read-only ListModel from an array of objects.

Example program to illustrates JList

```
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:



JCombo Box

Swing provides a combo box (a combination of a text field and a drop-down list) through the JComboBox class. A combo box normally displays one entry, but it will also display a drop-down list that allows a user to select a different entry.

The JComboBox constructor used by the example is shown here:

JComboBox(Object[ ] items)

Here, items is an array that initializes the combo box. Other constructors are available.JComboBox uses the ComboBoxModel. Mutable combo boxes (those whose entries canbe changed) use the MutableComboBoxModel.

In addition to passing an array of items to be displayed in the drop-down list, items canbe dynamically added to the list of choices via the addItem( ) method, shown here:

void addItem(Object obj)

Here, obj is the object to be added to the combo box. This method must be used only withmutable combo boxes.

One way to obtain the item selected in the list is to call getSelectedItem( ) on the combobox. It is shown here:

Object getSelectedItem( )

You will need to cast the returned value into the type of object stored in the list.

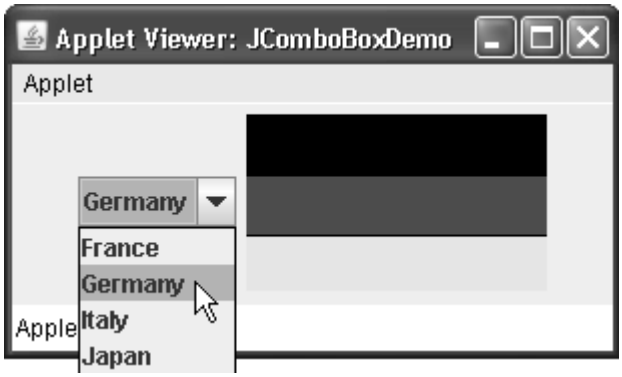
Example program to demonstrates the ComboBox

The following example demonstrates the combo box. The combo box contains entries for “France,” “Germany,” “Italy,” and “Japan.” When a country is selected, an icon-based label is updated to display the flag for that country. You can see how little code is required to use this powerful component.

```
// Demonstrate JComboBox.

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JComboBoxDemo" width=300 height=100>
```

```
</applet>
*/
public class JComboBoxDemo extends JApplet {
JLabel jlab;
ImageIcon france, germany, italy, japan;
JComboBox jcb;
String flags[] = { "France", "Germany", "Italy", "Japan" };
public void init() {
try {
SwingUtilities.invokeAndWait(
new Runnable() {
public void run() {
makeGUI();
}
}
);
} catch (Exception exc) {
System.out.println("Can't create because of " + exc);
}
}
private void makeGUI() {
// Change to flow layout.
setLayout(new FlowLayout());
// Instantiate a combo box and add it to the content pane.
jcb = new JComboBox(flags);
add(jcb);
// Handle selections.
jcb.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent ae) {
String s = (String) jcb.getSelectedItem();
jlab.setIcon(new ImageIcon(s + ".gif"));
}
});
// Create a label and add it to the content pane.
jlab = new JLabel(new ImageIcon("france.gif"));
add(jlab);
}
}
```

Output:



Swing Menus [2 marks]

JMenuBar, JMenu and JMenuItem

The JMenuBar class is used to display menubar on the window or frame. It may have several menus.
The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.
The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

JMenuBar class declaration

public class JMenuBar extends JComponent implements MenuElement, Accessible

JMenu class declaration

public class JMenu extends JMenuItem implements MenuElement, Accessible

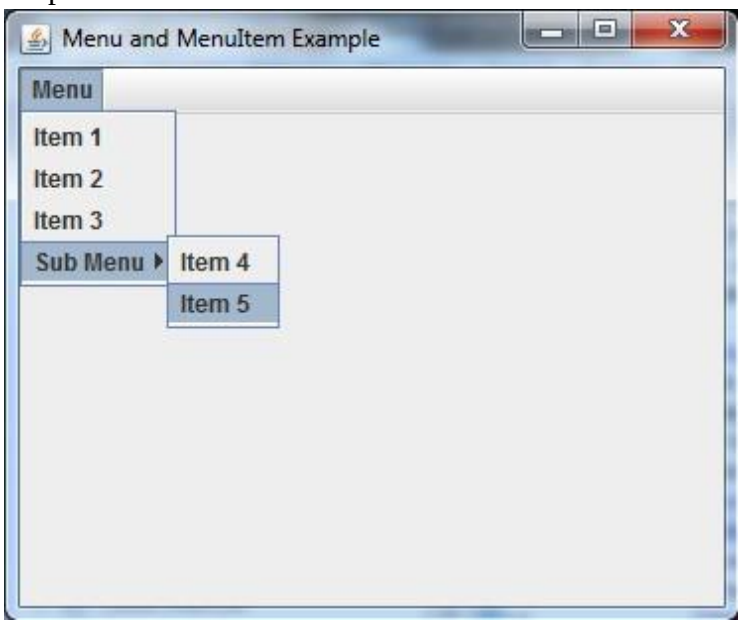
JMenuItem class declaration

public class JMenuItem extends AbstractButton implements Accessible, MenuElement

Example program to demonstrate SWING Menu

```
import javax.swing.*;
class MenuExample
{
        JMenu menu, submenu;
        JMenuItem i1, i2, i3, i4, i5;
        MenuExample(){
        JFrame f= new JFrame("Menu and MenuItem Example");
        JMenuBar mb=new JMenuBar();
        menu=new JMenu("Menu");
        submenu=new JMenu("Sub Menu");
        i1=new JMenuItem("Item 1");
        i2=new JMenuItem("Item 2");
        i3=new JMenuItem("Item 3");
        i4=new JMenuItem("Item 4");
        i5=new JMenuItem("Item 5");
        menu.add(i1); menu.add(i2); menu.add(i3);
        submenu.add(i4); submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setJMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
}
public static void main(String args[])
{
new MenuExample();
}}
```

Output:



Dialogs

The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box. These dialog boxes are used to display information or get input from the user. The JOptionPane class inherits JComponent class.

JOptionPane class declaration

public class JOptionPane extends JComponent implements Accessible

Common Constructors of JOptionPane class

Constructor	Description
JOptionPane()	It is used to create a JOptionPane with a test message.
JOptionPane(Object message)	It is used to create an instance of JOptionPane to display a message.
JOptionPane(Object message, int messageType	It is used to create an instance of JOptionPane to display a message with specified message type and default options.

Common Methods of JOptionPane class

Methods	Description
JDialog createDialog(String title)	It is used to create and return a new parentless JDialog with the specified title.
static void showMessageDialog(Component parentComponent, Object message)	It is used to create an information-message dialog titled "Message".
static void showMessageDialog(Component parentComponent, Object message, String title, int messageType)	It is used to create a message dialog with given title and messageType.
static int showConfirmDialog(Component parentComponent, Object message)	It is used to create a dialog with the options Yes, No and Cancel; with the title, Select an Option.
static String showInputDialog(Component parentComponent, Object message)	It is used to show a question-message dialog requesting input from the user parented to parentComponent.
void setInputValue(Object newValue)	It is used to set the input value that was selected or input by the user.

Example program to demonstrate showMessageDialog()

```
import javax.swing.*;

public class OptionPaneExample {

JFrame f;

OptionPaneExample(){

    f=new JFrame();

 JOptionPane.showMessageDialog(f,"Hello, Welcome to Java.");

}

public static void main(String[] args) {

    new OptionPaneExample();

}

}
```

Output:

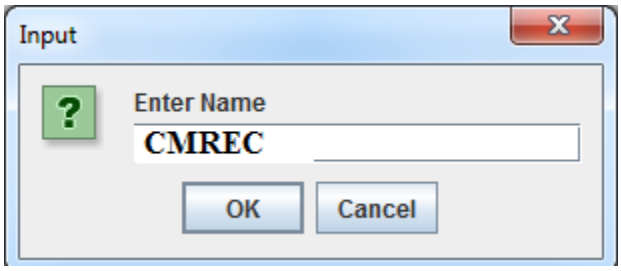
Example program to demonstrate showMessageDialog()

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    JOptionPane.showMessageDialog(f,"Successfully Updated.","Alert",JOptionPane.WARNING_MESSAGE);
}
public static void main(String[] args) {
    new OptionPaneExample();
}
}
```

Example program to demonstrate showInputDialog()

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
   f=new JFrame();
   String name=JOptionPane.showInputDialog(f,"Enter Name");
}
public static void main(String[] args) {
   new OptionPaneExample();
}
}
```

Output:



5.3.9 Difference between AWT and Swing [2 maks]

No.	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 and feel**.	Swing **supports pluggable look 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**.

Difference between Swing and Applet[3 marks]

Swing	Applet
Swing is light weight Component.	Applet is heavy weight Component.
Swing have look and feel according to user view you can change look and feel using UIManager.	Applet Does not provide this facility.
Swing uses for stand lone Applications, Swing have main method to execute the program.	Applet need HTML code for Run the Applet.
Swing uses MVC Model view Controller.	Applet not.
Swing have its own Layout like most popular Box Layout.	Applet uses AWT Layouts like flowlayout.
Swing have some Thread rules.	Applet doesn't have any rule.
To execute Swing no need any browser By which we can create stand alone application But Here we have to add container and maintain all action control with in frame container.	To execute Applet programe we should need any one browser like Appletviewer, web browser. Because Applet using browser container to run and all action control with in browser container.

UNIT_V JNTUH PREVIOUSLY ASKEDSHORT ANSWER QUESTIONS

1. What are the containers available in swing? [2M]
2. Compare Applets with application programs. [3M]
3. what is the use of LayOut manager
4. explain about life cycle of Applet
5. What are the merits of swing components over AWT? [2]
6. What is an adapter class? What is its significance? List the adapter classes. [3]
7. What are the sources for item event? [2]
8. Give the hierarchy for swing components. [3]
9. Give the subclasses of JButton class.
10. Differentiate between grid layout and border layout managers.
11. What are the limitations of AWT? [2]
12. Why do applet classes need to be declared as public? [3]
13. Give the AWT hierarchy. [2]
14. What are the various classes used in creating a swing menu? [3]
15. What are the differences between JToggle buttion and Radio buttion? [2]
16. What is an adapter class? Explain with an example. [3]
17. What is Swing in Java? How it differs from Applet. [2]
18. How do applets differ from application program? [3]

UNIT-V JNTUH-PREVIOUSLY ASKD ESSAY QUESTIONS

1a) What is the significance of layout managers? Discuss briefly various layout managers.
b) Give an overview of JButton class. [5+5]

2a) Explain delegation event model.
b) Write an Applet to draw a smiley picture accept user name as a parameter and display welcome message. [5+5]

3a) Describe about various components in AWT.
b) Write an applet program to handle all mouse events. [5+5]

4a) Write a Java program to create AWT radio buttons using check box group.
b) Explain the various event listener interfaces. [5+5]

5 a)Write a java program that design scientific calculator using AWT
b)What are the different types of Event listeners supported by java

6 a) Is Applet more secure than application program? Justify your answer.
b) Design a user interface to collect data from the student for admission application using swing components. [5+5]

7. Write a program to demonstrate various keyboard events with suitable functionality. [10]

8.a) Why swing components are preferred over AWT components?
b) What is an adapter class? What is their role in event handling? [5+5]

9a) Explain the life cycle of an applet.
b) What are the various layout managers used in Java? [5+5]

10.a) What is the role of event listeners in event handling? List the Java event listeners
b) Write an applet to display the mouse cursor position in that applet window.[5+5]

11.a) Discuss various AWT containers with examples.
b) Explain about the adapter class with an example.[5+5]

12.a) What is an applet? Explain the life cycle of Applet with a neat sketch.
b) Write the applets to draw the Cube and Cylinder shapes. [5+5]

13.a) What is an Layout manager? Explain different types of Layout managers.
b) Write a program to create a frame window that responds to key strokes. [5+5]

14.a) Illustrate the use of Grid Bag layout.
b) What are the subclasses of JButton class of swing package? [5+5]

15.a) Create a simple applet to display a smiley picture using Graphics class methods.
b) Write a short note on delegation event model. [5+5]

16 a) List and explain different types of Layout managers with suitable examples.
b) How to move/drag a component placed in Swing Container? Explain. [5+5]

17.a) Discuss about different applet display methods in brief.
b) What are the various components of Swing? Explain. [5+5]

18 a) What is the difference between init( ) and start ( ) methods in an Applet? When will each be executed?
b) Write the applets to draw the Cube and Circle shapes. [5+5]

19. a) Explain various layout managers in JAVA.
b) Write a program to create a frame window that responds to mouse clicks. [5+5]