Unit-I

Topics:

Event Handling | Abtract Window Toolkit

Q1. Explain the delegation event model in event handling.

- Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has the code which is known as event handler that is executed when an event occurs.
- Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.
- The Delegation Event Model has the following key participants namely:

  Events: In the delegation model, an event is an object that describes a state change in a source. It can be 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.

  Source: The source is an object on which event occurs. Source is responsible for providing information of the occurred event to its handler. Java provide as with classes for source object. 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**( ).

  Listener: It is also known as event handler. Listener is responsible for generating response to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received, the listener processes the event and then returns.

- The benefit of this approach is that the user interface logic is completely separated from the logic that generates the event. The user interface element is able to delegate the processing of an event to the separate piece of code

Q2 List various the various Event classes and Event listener.

Event classes

Table enumerates the most important of these event classes and provides a brief description of when they are generated.

Event Class	Description
ActionEvent	Generated when a button is pressed, a list is double-clicked, or a menuitem is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
FocusEvent	Generated when a component gains or loses keyboard focus.
ItemEvent	Generated when a check box or a list item is clicked; also occurs when achoice selection is made or a checkable menu is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.

MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window os activated, closed, deactivated, deiconified,iconified, opened, or quit.

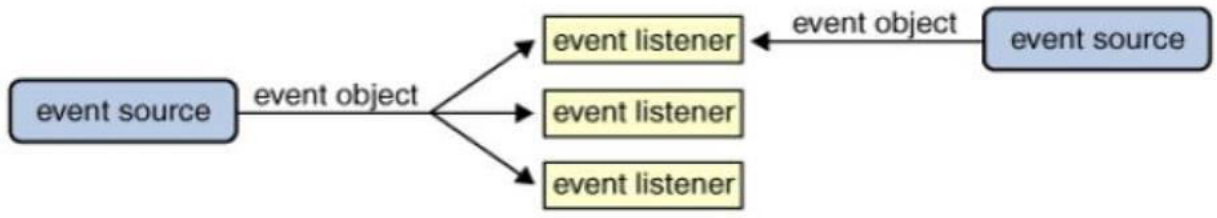
Event Listener

Following Table lists commonly used listener interfaces and provides a brief description of the methods that they define.

Interface	**Description**
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
TextListener	Defines one method to recognize when a text value changes.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Q3. Write a short note on "Event Listeners". Explain the working with code specification.

The event model is quite powerful and flexible. Any number of event listener objects can listen for all kinds of events from any number of event source objects. For example, a program might create one listener per event source or a program might have a single listener for all events from all sources. A program can even have more than one listener for a single kind of event from a single event source.



Multiple listeners can register to be notified of events of a particular type from a particular source. Also, the same listener can listen to notifications from different objects.
Each event is represented by an object that gives information about the event and identifies the event source.

```
class Abc extends Frame implements ActionListener
{
Button ok,cancel;
TextBox t1;
Abc()
{
t1=new TextBox();
ok=new Button(“OK”);
cancel=new Button(“CANCEL”);
ok.addActionListener(this);
cancel.addActionListener(this);
add(ok);
add(cancel);
add(t1);
}

Public void actionPerformed(ActionEvent ae)
{
if(ae.getActionCommand()==”OK”)
t1.setText(“ You clicked OK button”) ;
else if(ae.getActionCommand()==”CANCEL”)
t1.setText(“You clicked Cancel button”);
}
public static void main(String arg[])
{
new Abc();}
}
}
```

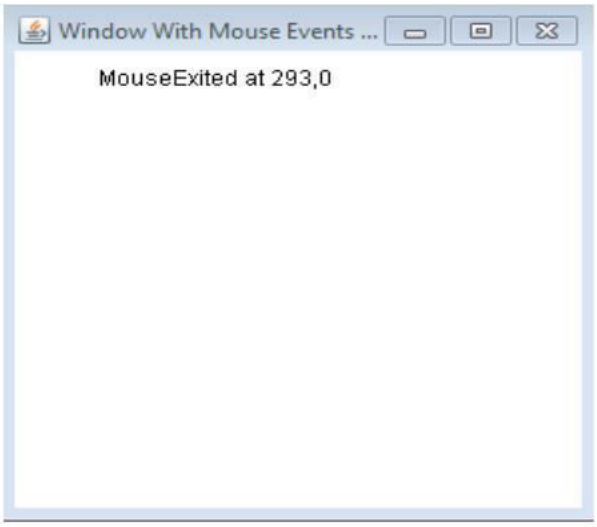
Q4. Write a java program to handle the mouse related events.

Example:

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

public class MouseListenerEx extends Frame implements MouseListener
{
int x=0, y=0; String str= "";

MouseListenerEx (String title)
{
super(title);
addWindowListener(new MyWindowAdapter(this));
addMouseListener(this);
setSize(300,300);
setVisible(true);
}

public void mouseClicked(MouseEvent e)
{
```

```
str= "MouseClicked";
x = e.getX();
y = getY();
repaint();
}

public void mousePressed(MouseEvent e)
{
str = "MousePressed";
x = e.getX();
y = getY();
repaint();
}

public void mouseReleased(MouseEvent e)
{
str = "MouseReleased";
x = e.getX();
y = getY();
repaint();
}

public void mouseEntered(MouseEvent e)
{
str= "MouseEntered";
x = e.getX();
y = getY();
repaint();
}

public void mouseExited(MouseEvent e)
{
str = "MouseExited";
x = e.getX();
y = getY();
repaint();
}

public void paint(Graphics g)
{
g.drawString(str + " at " + x + "," + y, 50,50);
}

public static void main(String[] args)
{
MouseListenerEx  m= new MouseListenerEx ("Window With Mouse Events Example");
}
}

class MyWindowAdapter extends WindowAdapter
{
```

```
MouseListenerEx myWindow = null;
MyWindowAdapter(MouseListenerEx myWindow)
{
this.myWindow = myWindow;
}
public void windowClosing(WindowEvent we)
{
myWindow.setVisible(false);
}
}
```



Q5. Write a short note on AWT

The **Abstract Window Toolkit** (AWT) is class library provides a user interface toolkit called the Abstract Windowing Toolkit, or the AWT.

The AWT is both powerful and flexible. At the lowest level, the operating system transmits information from the mouse and keyboard to the program as input, and provides pixels for program output.

The AWT provides a well-designed object-oriented interface to these low-level services and resources. Because the Java programming language is platform-independent, the AWT must also be platform-independent. The AWT was designed to provide a common set of tools for graphical user interface design that work on a variety of platforms.

The AWT classes are contained in the java.awt package. It is one of Java's largest packages. We will list some of the AWT classes:

Class	**Description**
Button	Creates a push button control.
Checkbox CheckboxGroup	Creates a check box control. Creates a group of check box controls.
Choice	Creates a pop-up list.
Color	Manages colors in a portable, platform-independent fashion.
FlowLayout	The flow layout manager. Flow layout positions components left to right, top to bottom.
Frame	Creates a standard window that has a title bar, resize corners, and

	a menu bar.
GridLayout	The grid layout manager. Grid layout displays components in a two-dimensional grid.
Label	Creates a label that displays a string.
List	Creates a list from which the user can choose. Similar to the standard Windows list box.
ScrollPane	A container that provides horizontal and/or vertical scroll bars for another component.
TextArea	Creates a multiline edit control.
TextField	Creates a single-line edit control.

Q6. What are different operations that can be carried out on Frame Window?

The class Frame is a top level window with border and title. It uses BorderLayout as default layout manager.

Using below method various operation can be performed.

Method	Description
setVisisble()	Sets whether this frame is visible to user or not.
setSize()	Sets the size of frame in width and height
setTitle()	Sets the title for this frame to the specified string.
setResizable()	Sets whether this frame is resizable by the user.
setMenuBar()	Sets the menu bar for this frame to the specified menu bar.

Example:

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

public class FrameDemo extends  Frame
{
Label l;
public FrameDemo()
{
setLayout(new FlowLayout());
l=new Label(“Hello World”);
setTitle("Frame Demo");
setSize(400,300);
add(l);
setVisible(true);
}
}
public static void main(String ar[])
{
FrameDemo f=new FrameDemo();
}

}
```

Q7. With respect to AWT explain following controls:

1. **Button**
2. **Checkbox**
3. **Choice**
4. **List**
5. **Scrollbar**

Button

Button is a control component that has a label and generates an event when pressed. When a button is pressed and released, AWT sends an instance of ActionEvent to the button, by calling processEvent on the button.

Below example demonstrate use of Button in AWT:

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

public class ButtonDemo extends  Frame
{
Button b;
public ButtonDemo()
{
setLayout(new FlowLayout());
b=new Button(“Click Here”);
setTitle("Button Demo");
setSize(400,300);
add(b);
setVisible(true);
}
}
public static void main(String ar[])
{
ButtonDemo f=new ButtonDemo();
}

}
```

Checkbox

Checkbox control is used to turn an option on(true) or off(false). There is label for each checkbox representing what the checkbox does.The state of a checkbox can be changed by clicking on it.

Below example demonstrate use of Checkbox in AWT:

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

public class CheckboxDemo extends  Frame
{
```

```
Checkbox c1,c2,c3;
Label l;
public CheckboxDemo ()
{
setLayout(new FlowLayout());
l=new Label(“Select Your Hobbies:”);
c1=new Checkbox(“Cricket”);
c2=new Checkbox(“Carrom”);
c3=new Checkbox(“Football”);
setTitle("Checkbox Demo");
setSize(400,300);
add(l);
add(c1);
add(c2);
add(c3);
setVisible(true);
}
}
public static void main(String ar[])
{
CheckboxDemo f=new CheckboxDemo();
}

}
```

Choice

Choice control is used to show pop up menu of choices. Selected choice is shown on the top of the menu.

Below example demonstrate use of Choice in AWT:

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

public class ChoiceDemo extends  Frame
{
Choice ddl;
Label l;
public ChoiceDemo ()
{
setLayout(new FlowLayout());
l=new Label(“Select State:”);
ddl=new Choice();
ddl.add(“Maharashtra”);
ddl.add(“Uttar Pradesh”);
ddl.add(“Goa”);
ddl.add(“Gujrat”);
setTitle("Choice  Demo");
setSize(400,300);
add(l);
add(ddl);
```

```
setVisible(true);
}
}
public static void main(String ar[])
{
ChoiceDemo f=new ChoiceDemo ();
}

}
```

List

The List represents a list of text items. The list can be configured to that user can choose either one item or multiple items.

Below example demonstrate use of List in AWT:

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

public class ListDemo extends  Frame
{
List lst;
Label l;
public  ListDemo()
{
setLayout(new FlowLayout());
l=new Label(“Select  Hobbies:”);
lst=new List(4,true);
lst.add(“Cricket”);
lst.add(“Football”);
lst.add(“Music”);
lst.add(“Singing”);
setTitle("List  Demo");
setSize(400,300);
add(l);
add(lst);
setVisible(true);
}
}
public static void main(String ar[])
{
ListDemo f=new ListDemo ();
}

}
```

Scrollbar

Scrollbar control represents a scroll bar component in order to enable user to select from range of values.

Below example demonstrate use of Scrollbar in AWT:

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

public class ScrollbarDemo extends  Frame
{
public  ScrollbarDemo ()
{
setLayout(new FlowLayout());
Scrollbar horizontalScroller = new Scrollbar(Scrollbar.HORIZONTAL);
Scrollbar verticalScroller = new Scrollbar();
verticalScroller.setOrientation(Scrollbar.VERTICAL);
horizontalScroller.setMaximum (100);
horizontalScroller.setMinimum (1);
verticalScroller.setMaximum (100);
verticalScroller.setMinimum (1);
 setTitle("Scrollbar  Demo");
setSize(400,300);
add(horizontalScroller);
add(verticalScroller);
setVisible(true);
}
}
public static void main(String ar[])
{
ScrollbarDemo f=new ScrollbarDemo ();
}

}
```

Q8. How does AWT create Radio Buttons? Explain with syntax and code specification.

OR

What is CheckBoxGroup? Explain with Example.

The java AWT, top-level window, is represented by the CheckBoxGroup. In this program, we will see how to create and show the CheckboxGroup component on the frame.

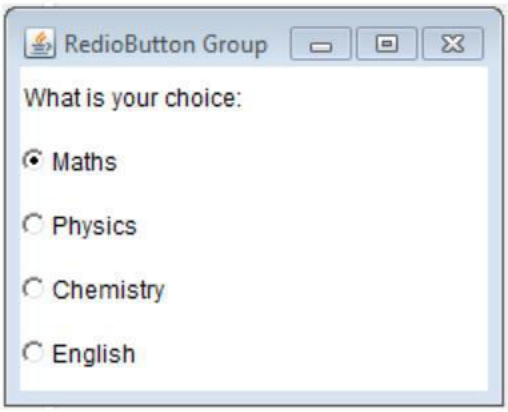
In below program a radio button is created that is an item that can be selected or deselected and displays that state to the user. Here we are creating a group of buttons in which we can select only one option at a time.

```
CheckBoxGroup object=new CheckBoxGroup ();
```

```
import java.awt.*;
import java.awt.event.*;
public class RadioButtonEx{
public static void main(String[] args)
{
Frame fm=new Frame("RedioButton Group");
Label la=new Label("What is your choice:");
fm.setLayout(new GridLayout(0, 1));
```

```
CheckboxGroup cg1=new CheckboxGroup();
fm.add(la);
fm.add(new Checkbox("Maths", cg1, true));
fm.add(new Checkbox("Physics", cg1, false));
fm.add(new Checkbox("Chemistry", cg1, false));
fm.add(new Checkbox("English", cg1, false));
fm.setSize(250,200);
fm.setVisible(true);
fm.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we){
System.exit(0);
}
});
}
}
```

Output:



Q9. What are adapter classes? Why are they used?

- An adapter class provides the default implementation of all methods in an event listener interface. An adapter class provides an empty implementation of all methods in an event listener interface i.e. this class itself write definition for methods which are present in particular event listener interface.
- Adapter classes are very useful when we want to process only few of the events that are handled by a particular event listener interface.
- We can define a new class by extending one of the adapter classes and implement only those events relevant to us.
- "Every listener that includes more than one abstract method has got a corresponding adapter class". The advantage of adapter is that we can override any one or two methods we like instead of all.
- Adapter classes are useful when we want to receive and process only some of the events that are handled by a particular event listener interface.

For Example

The **MouseMotionAdapter** class has 2 methods, **mouseDragged**( ) &**mouseMoved**( ), which are the methods defined by the MouseMotionListener interface.
If we were interested in only mouse drag events, then we could simply extend MouseMotionAdapter and override mouseDragged( ).

The empty implementation of mouseMoved( ) would handle the mouse motion events for us. Some of the Adapter classes are mentioned below.

Adapter Classes
KeyAdpater
MouseAdapter
MouseMotionAdapter
WindowAdapter

Example:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/*
<applet code="TAdapterDemo" width=300 height=100>
</applet>
*/
public class TAdapterDemo extends Applet
{
      public void init( ){
      addMouseListener(
      new MouseAdapter( ){
      int X, Y;
      public void mouseReleased(MouseEvent me){
      X = me.getX( );
      Y = me.getY( );
      Graphics g = TAdapterDemo.this.getGraphics();
      g.drawString("Mouse Realeased", X,Y);
    }
    });
   }
}
```

Q10. What are inner classes? Explain with example

In object-oriented programming, an inner class or nested class is a class declared entirely within the body of another class or interface. It is distinguished from a subclass. So an inner class is a class defined within another class, or even within an expression.

Inner classes can be used to simplify the code when using event adapter classes as shown below.

Example:

```
class Outer
{
      int outer_x = 100;
      void test()
     {
      Inner inner = new Inner();
      inner.display();
      }

     class Inner
```

```
    {
      void display()
      {
          System.out.println("display: outer_x = " + outer_x);
       }
      }
}
class InnerClassDemo
{
      public static void main(String args[])
{
      Outer outer = new Outer();
      outer.test();
}
}
```

Q11. What are anonymous classes? Explain with example

An anonymous inner class is one that is not assigned a name. This section illustrates how an anonymous inner class can facilitate the writing of event handlers. Consider the applet shown in the following listing. As before, its goal is to display the string "Mouse Pressed" in the status bar of the applet viewer or browser when the mouse is pressed.

Example

```
import java.applet.*;
import java.awt.event.*;
/*
<applet code="Abc" width=200 height=100>
</applet>
*/
public class Abc extends Applet {
public void init() {
addMouseListener(new MouseAdapter() {
public void mousePressed(MouseEvent me) {
showStatus("Mouse Pressed");
}
});
}
}
```

Q12. What are the different Layout Managers classes in java? Explain Card Layout With Example.

This class support following various types of layout as follows.

LayoutManager	Description
BorderLayout	The BorderLayout arranges the components to fit in the five regions: east, west, north, south and center.
CardLayout	The CardLayout object treats each component in the container as a card. Only one card is visible at a time.
FlowLayout	The FlowLayout is the default layout. It layouts the components

	in a directional flow.
GridLayout	The GridLayout manages the components in form of a rectangular grid.

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

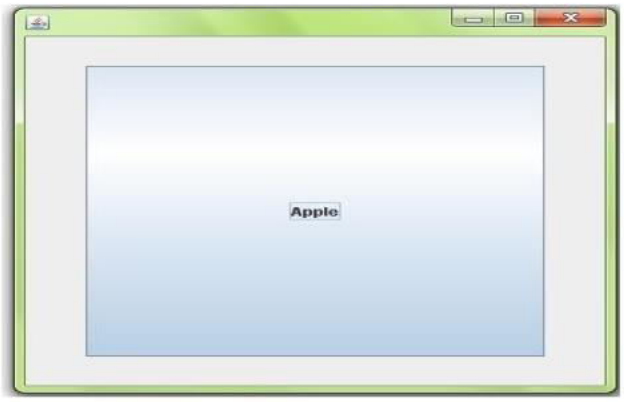
Commonly used **methods** of CardLayout class:
public void **next**(Container parent): is used to flip to the next card of the given container.
public void **previous**(Container parent): is used to flip to the previous card of the given container.
public void **first**(Container parent): is used to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

Example:

```
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);
      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);
} }
```

Output:



GridLayout

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

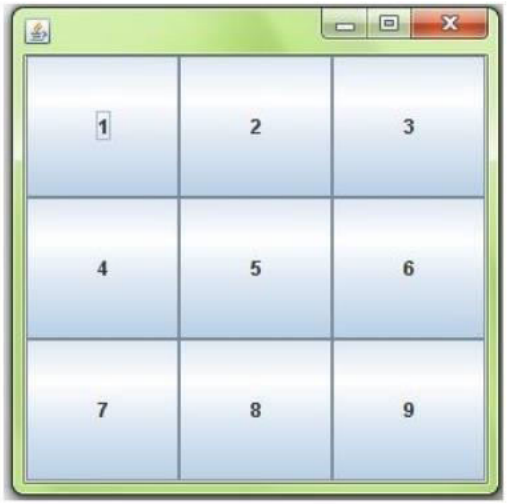
Constructors of GridLayout class:
GridLayout(): creates a grid layout with one column per component in a row.
GridLayout(int rows, int columns): creates a grid layout with the given rows and columns but no gaps between the components.
GridLayout(int rows, int columns, int hgap, int vgap): creates a grid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

Example:

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

public class GridLayoutEx{
JFrame f;
GridLayoutEx(){
    f=new JFrame();

    JButton b1=new JButton("1");
    JButton b2=new JButton("2");
    JButton b3=new JButton("3");
    JButton b4=new JButton("4");
    JButton b5=new JButton("5");
    JButton b6=new JButton("6");
    JButton b7=new JButton("7");
```

```
    JButton b8=new JButton("8");
       JButton b9=new JButton("9");

    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
    f.add(b6);f.add(b7);f.add(b8);f.add(b9);

    f.setLayout(new GridLayout(3,3));
    //setting grid layout of 3 rows and 3 columns

    f.setSize(300,300);
    f.setVisible(true);
}
public static void main(String[] args) {
    new GridLayoutEx();
}
}
```

Output:



FlowLayout

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.

Fields of FlowLayout class:
public static final int LEFT
public static final int RIGHT
public static final int CENTER
public static final int LEADING
public static final int TRAILING

Constructors of FlowLayout class:
FlowLayout(): creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
FlowLayout(int align): creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.

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

Example:

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

public class FlowLayoutEx
{
JFrame f;
FlowLayoutEx(){
   f=new JFrame();

   JButton b1=new JButton("1");
   JButton b2=new JButton("2");
   JButton b3=new JButton("3");
   JButton b4=new JButton("4");
   JButton b5=new JButton("5");

   f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);

   f.setLayout(new FlowLayout(FlowLayout.RIGHT));

   f.setSize(300,300);
   f.setVisible(true);
}
public static void main(String[] args) {
   new FlowLayoutEx();
}
}
```

Output:

Q13. Explain the Border layout used in java with the help of a program.

OR

What is the default layout of the Frame? Explain the same.

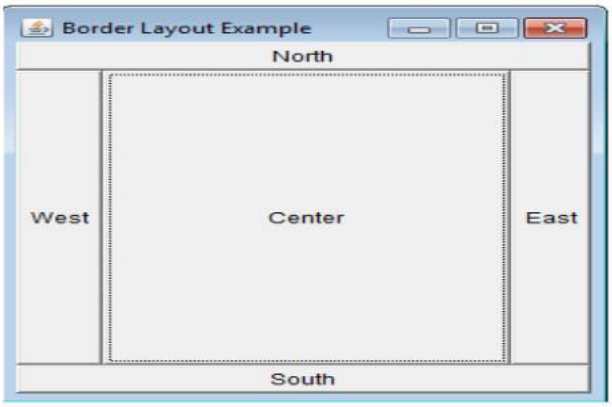
BorderLayout is the default layout manager for all Windows, such as Frames and Dialogs. It uses five areas to hold components: north, south, east, west, and center.

All extra space is placed in the center area. When adding a component to a container with a border layout, use one of these five constants.

Example:

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

public class BorderLayoutDemo extends Frame
{
Button b1,b2,b3,b4,b5;
public BorderLayoutDemo()
{
BorderLayout  b=new BorderLayout();
setLayout(b);
b1=new Button("Center");
                                        b2=new Button("South");
b3=new Button("North");
b4=new Button("East");
b5=new Button("West");
setTitle("Border Layout Example");
add(b1,BorderLayout.CENTER);
add(b2,BorderLayout.SOUTH);
add(b3,BorderLayout.NORTH);
add(b4,BorderLayout.EAST);
add(b5,BorderLayout.WEST);
setSize(500,500);
setVisible(true);
}
public static void main(String ar[])
{
BorderLayoutDemo  bl=new BorderLayoutDemo();
}

}
```

Output:



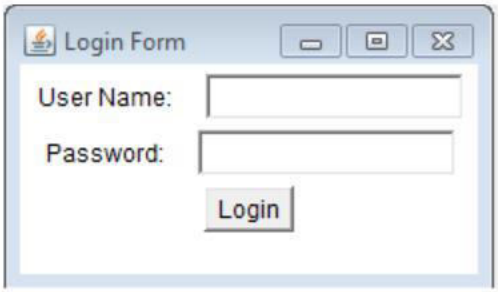
Q14. Write a java AWT program that creates the following GUI.

Code Sample:

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

```
public class Login extends Frame
{

Label l1,l2;
TextField txtUser,txtPwd;
Button b;
public Login()
{
l1=new Label("User Name:");
l2=new Label("Password:");
txtUser=new TextField(15);
txtPwd=new TextField(15);
txtPwd.setEchoChar('#');
b=new Button("Login");

setLayout(new FlowLayout());
setSize(300,300);
setTitle("Login Form");
add(l1);
add(txtUser);
add(l2);
add(txtPwd);
add(b);
setVisible(true);
}

public static void main(String ar[])
{
Login l=new Login();
}
}
```

Output:



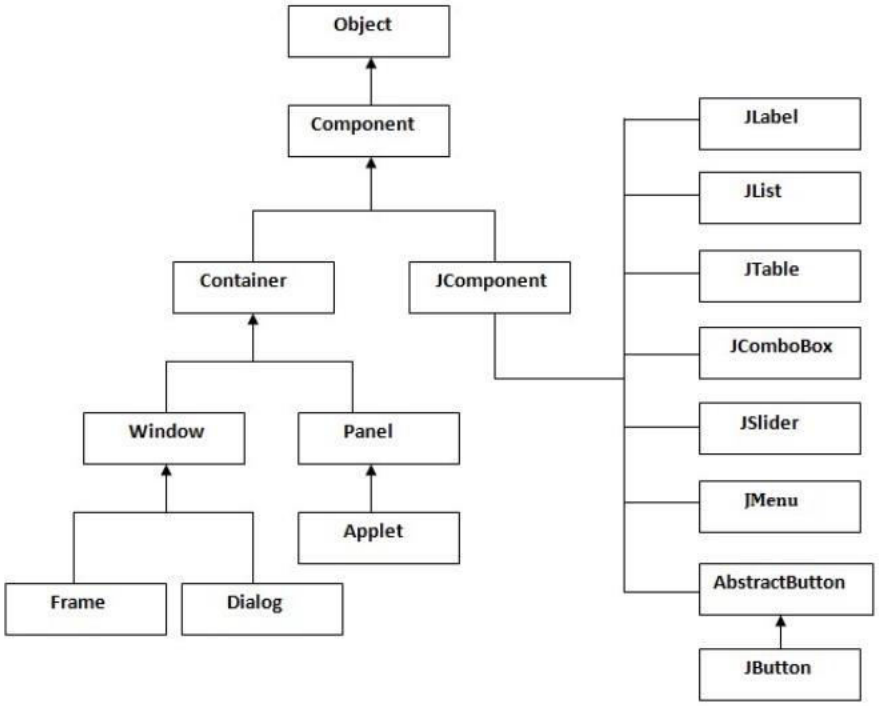
Unit-II

Topics:

Swing

Q1. What are swings in java? Explain the features of swing

- Swing/JFC is short for **Java Foundation Classes**, which encompass a group of features for building graphical user interfaces (GUIs) and adding rich graphics functionality and interactivity to Java applications.
- The swing API contains 100% pure java versions of the AWT components, plus many additional components that are also 100% pure java, because swing does not contain or depend on native code.
- Swing is the package or set of classes that provide more powerful and flexible component than that of 'AWT'.
- It supplies several exiting addition including tabbed panes, trees and tables.
- Even familiar component like buttons have more capabilities in swing i.e. buttons may have both image and text associated with it.
- The hierarchy of java swing API is given below.



Following are features of swing.

Swing Components Are Lightweight

This means that they are written entirely in Java and do not map directly to platform specific peers. Because lightweight components are rendered using graphics primitives, they can be transparent which enables non rectangular shapes. Thus, lightweight components are more efficient and more flexible.

Swing Supports a Pluggable Look and Feel

Each Swing component is rendered by Java code rather than by native peers, the look and feel of a component is under the control of Swing.

It is possible to "plug in" a new look and feel for any given component without creating any side effects in the code that uses that component.

The MVC Architecture

This is Essential feature of Swing that is using MVC we can change internal representation of table, trees and list.

New Components

There are various new components provided by swing which are as follows

Image Button
Tabbed Panes
Sliders
Toolbars
Text Areas
List
Trees
Tables

Q2. Explain the new features of JFC.

The JFC (Java Foundation Classes) are a set of GUI components and services which simplify the development and deployment of commercial quality of desktop and Internet or Intranet applications.

Its features are:

- Java Foundation classes are core to the Java 2 Platform.
- All JFC components are JavaBeans components and therefore reusable, interoperable, and portable.
- JFC offers an open architecture. Third-party JavaBeans components can be used to enhance applications written using JFC.
- It is truly cross-platform.

Q3. Compare Swing and AWT.

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

A.W.T	Swing
AWT components are platform-dependent.	Java swing components are platform-independent
AWT components are heavyweight.	Swing components are lightweight
AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.
AWT provides less components than Swing	Swing provides more powerful components such as tables, lists.
AWT doesn't follow MVC architecture.	Swing follows MVC architecture.
It does not consist of new components.	It consists of new components such as Sliders, Tabbed Button, Toolbars and Tables and so on
AWT is a thin layer of code on top of the OS. Controls does not start with character J.	Swing is much larger. Swing also has very much richer functionality. In swing Controls start with character J. **Ex**. JButton , JTextFiled

Q12. Write a java program using JCheckBox and JRadioButton.

JRadioButton

- 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.

Constructor of JRadioButton:

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.

Example of JRadioButton with source code

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

class Xyz extends JFrame
{
JPanel jpl;
JLabel lbl;
JRadioButton rnd1,rnd2;
ButtonGroup bgrp;
Xyz()
{
jpl=new JPanel();
bgrp=new ButtonGroup();
lbl=new JLabel("Select Job Type:");
rnd1=new JRadioButton("Full Time");
rnd2=new JRadioButton("Part Time");
bgrp.add(rnd1);
bgrp.add(rnd2);
setTitle("JCheckbox Example");
setSize(300,300);
jpl.add(lbl);
jpl.add(rnd1);
jpl.add(rnd2);
add(jpl);
setVisible(true);
}

}

class JRadioButtonEx
{
public static void main(String ar[])
{
Xyz x1=new Xyz();
}
}
```

JCheckBox

The JCheckBox class is used to create the checkbox and use can select multiple option.

Constructor of JCheckBox:

JCheckBox (): creates an unselected checkbox with no text.
JCheckBox (String s): creates an unselected checkbox with specified text.

Example of JCheckBox with source code

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

class Xyz extends JFrame
{
JPanel jpl;
JCheckBox chk1,chk2;
Xyz()
{
jpl=new JPanel();
chk1=new JCheckBox("Cricket");
chk2=new JCheckBox("Basketball");
setTitle("JCheckbox Example");
setSize(300,300);
jpl.add(chk1);
jpl.add(chk2);
add(jpl);
setVisible(true);
}

}

class JCheckboxEx
{
public static void main(String ar[])
{
Xyz x1=new Xyz();
}
}
```

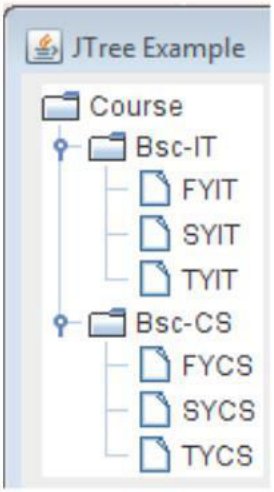
Q4. Write a java SWING program that creates the JTree GUI.

- JTree is a Swing component with which we can display hierarchical data. JTree is quite a complex component.
- A JTree has a 'root node' which is the top-most parent for all nodes in the tree. A node is an item in a tree. A node can have many children nodes.
- These children nodes themselves can have further children nodes. If a node doesn't have any children node, it is called a leaf node.

Example of JTree with source code

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

class Abc extends JFrame
{

JTree tree;
JPanel jpl;
public Abc()
{
DefaultMutableTreeNode r=new DefaultMutableTreeNode("Course");
DefaultMutableTreeNode i=new DefaultMutableTreeNode("Bsc-IT");
DefaultMutableTreeNode cs=new DefaultMutableTreeNode("Bsc-CS");
r.add(i);
r.add(cs);

DefaultMutableTreeNode fi=new DefaultMutableTreeNode("FYIT");
DefaultMutableTreeNode si=new DefaultMutableTreeNode("SYIT");
DefaultMutableTreeNode ti=new DefaultMutableTreeNode("TYIT");
i.add(fi);
i.add(si);
i.add(ti);

DefaultMutableTreeNode fcs=new DefaultMutableTreeNode("FYCS");
DefaultMutableTreeNode scs=new DefaultMutableTreeNode("SYCS");
DefaultMutableTreeNode tcs=new DefaultMutableTreeNode("TYCS");
cs.add(fcs);
cs.add(scs);
cs.add(tcs);

tree=new JTree(r);
jpl=new JPanel();
jpl.add(tree);
add(jpl);
setTitle("JTree Example");
setSize(500,500);
setVisible(true);
}

class JTreeEx
{
public static void main(String ar[])
{
Abc a1=new Abc();
}
}
```

Output:



Q5. Write a java program to create different tabs using JTabbedPane.

A JTabbedPane contains a tab that can have a tool tip and a mnemonic, and it can display both text and an image. With the JTabbedPane class, we can have several components, such as panels, share the same space. The user chooses which component to view by selecting the tab corresponding to the desired component

Example of JTabbedPane with source code

```
import javax.swing.*;

class Abc extends JFrame
{

JPanel jpl;
JTabbedPane jtp;
JButton b1,b2;

Abc()
{
jpl=new JPanel();
b1=new JButton("Button1");
b2=new JButton("Button2");
jtp=new JTabbedPane(2);
jtp.add("Tap1",b1);
jtp.add("Tap2",b2);
jpl.add(jtp);
setTitle("JFrame Example");
setSize(300,300);
add(jpl);
setVisible(true);
}

}

class JTappedPaneEx
{
```

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

Abc a1=new Abc();

}
}
```

Output:

Q6. Explain JPopupMenu Class with Example

Popup menu represents a menu which can be dynamically popped up at a specified position within a component.

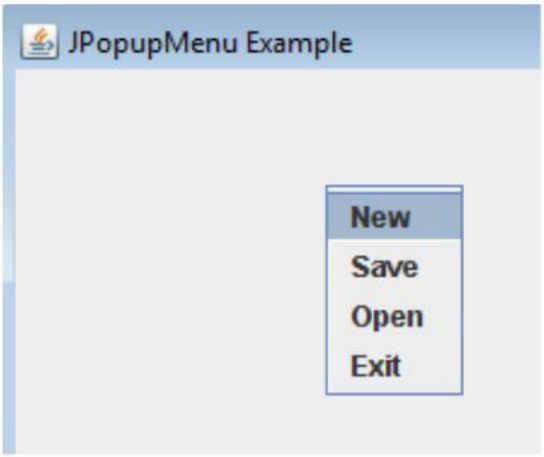
JPopupMenu Constructor
JPopupMenu (): Constructs a JPopupMenu without an "invoker".
JPopupMenu (String label): Constructs a JPopupMenu with the specified title.

Example of JPopupMenu with source code

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

class Abc extends JFrame
{

JPopupMenu jpm;
JMenuItem i1,i2,i3,i4;
public Abc()
{
jpm=new JPopupMenu();
i1=new JMenuItem("New");
i2=new JMenuItem("Save");
i3=new JMenuItem("Open");
i4=new JMenuItem("Exit");
jpm.add(i1);
jpm.add(i2);
jpm.add(i3);
jpm.add(i4);
setTitle("JPopupMenu Example");
addMouseListener(new MouseAdapter()
{
public void mouseReleased(MouseEvent  me)
{
jpm.show(me.getComponent(),me.getX(),me.getY());
}
}
);
setSize(500,500);
setVisible(true);
```

```
}

}

class JPopupMenuEx
{
public static void main(String ar[])
{
Abc a1=new Abc();
}
}
```

Output:



Q7. How are menus created in java? What are the classes used for creating menus? Explain with example.

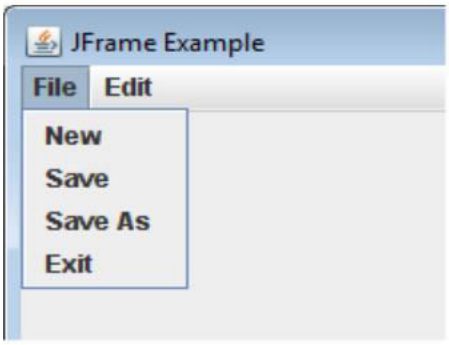
The JMenuBar class provides an implementation of a menu bar. For creating menu we use JMenuItem and JMenu classes.

Example of JMenu with source code

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

class Abc extends JFrame
{

JMenuBar jmb;
JMenu m1,m2;
JMenuItem i1,i2,i3,i4;
Abc()
{
jmb=new JMenuBar();
m1=new JMenu("File");
m2=new JMenu("Edit");
jmb.add(m1);
jmb.add(m2);
```

```
i1=new JMenuItem("New");
i2=new JMenuItem("Save");
i3=new JMenuItem("Save As");
i4=new JMenuItem("Exit");

m1.add(i1);
m1.add(i2);
m1.add(i3);
m1.add(i4);

setLayout(new FlowLayout());
setTitle("JFrame Example");
setSize(300,300);
setJMenuBar(jmb);
setVisible(true);
}

}
class JMenuBarEx
{
public static void main(String ar[])
{
Abc a1=new Abc();
}
}
```

Output:



Q8. Explain JScrollPane and JScrollBar with example.

JScrollPane

- It provides a scrollable view of a lightweight component. A JScrollPane manages a viewport, optional vertical and horizontal scroll bars, and optional row and column heading viewports.
- JScrollPane does not support heavyweight components.

JScrollBar

- The JScrollBar lets the user graphically select a value by sliding a knob within a bounded interval.
- It is us a very useful component when we want to display large amount of elements on the screen.

Constructors:

JScrollBar ( ): It Creates a JScrollBar instance with a range of 0-100, an initial value of 0, and vertical orientation.
JScrollBar (int Orientation): It creates a JScrollBar instance with a range of 0-100, an initial value of 0, and the specified orientation.

Methods:

getValue ( ) : It gives the current value of scroll bar.
setMaximum (int Max) :The maximum value of the scrollbar is maximum – extent.
setMinimum (int Min) : Returns the minimum value supported by the scrollbar (usually zero).
setValue (int Value) :Sets the scrollbar's value.

Example of JScrollPane with source code

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class Xyz extends JFrame
{
JPanel jpl;
JLabel lbl;
JTextArea txtAdd;
JScrollPane jsp;
Xyz()
{
String str="Plase mention address here";
jpl=new JPanel();
lbl=new JLabel("Address:");
txtAdd=new JTextArea(str,6,16);
jsp=new JScrollPane(txtAdd);
setTitle("JCheckbox Example");
setSize(300,300);
jpl.add(lbl);
jpl.add(jsp);
add(jpl);
setVisible(true);
}

}

class JScrollPaneEx
{
public static void main(String ar[])
{
Xyz x1=new Xyz();
}
}
```

Q9. Explain JColorChooser with example.

The JColorChooser class is used to create a color chooser dialog box so that user can select any color.

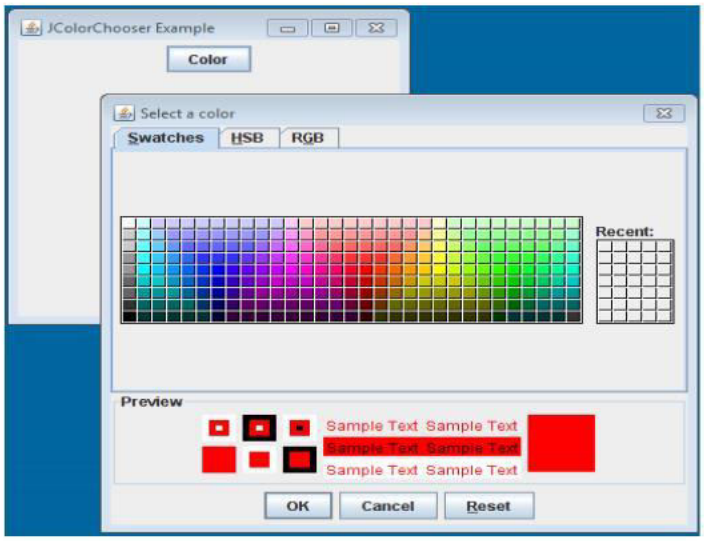
JPopupMenu Constructor

JColorChooser (): is used to create a color chooser pane with white color initially.
JColorChooser (Color c): is used to create a color chooser pane with the specified color initially.

Example of JColorChooser with source code

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

class Abc extends JFrame implements ActionListener
{

JButton b;
JPanel jpl;
Abc()
{
b=new JButton("Color");
b.addActionListener(this);
jpl=new JPanel();
setTitle("JColorChooser Example");
setSize(300,300);
jpl.add(b);
add(jpl);
setVisible(true);
}

public void actionPerformed(ActionEvent e)
{
Color initialcolor=Color.RED;
Color color=JColorChooser.showDialog(this,"Select a color",initialcolor);
setBackground(color);
}

}

class JColorChooserEx
{
public static void main(String ar[])
{
Abc a1=new Abc();
}
}
```

Output:



Q10. How do divide frame window in 2 parts? Explain with code specification.

The JSplitPane is commonly used component because it lets we want split our window horizontally or vertically in order to create a wide variety of GUI elements to suit our application's needs.

In short in order to create a JSplitPane component in Java, one should follow these steps:

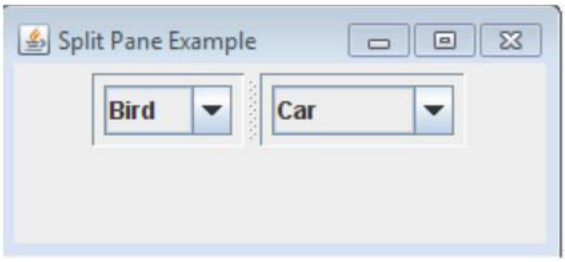
- Create a new JFrame.
- Call setLayout(new FlowLayout()) to set flow layout for the frame.
- Create two String arrays that will containt the contents of the two areas of theJSplitPane.
- Create two JScrollPane components.
- Create a new JSplitPane with the above JScrollPane components in each side.
- Use frame.getContentPane().add(splitPane) to add the spilt pane to our frame

Example of JMenu with source code

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

class Abc extends JFrame
{
JPanel p1,p2;
Abc()
{
String[] options1 = { "Bird", "Cat", "Dog", "Rabbit", "Pig" };
JComboBox combo1 = new JComboBox(options1);
String[] options2 = { "Car", "Motorcycle", "Airplane", "Boat" };
JComboBox combo2 = new JComboBox(options2);
p1 = new JPanel();
p1.add(combo1);
p2 = new JPanel();
p2.add(combo2);
JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, p1, p2);
setTitle("Split Pane Example");
setSize(200, 200);
setLayout(new FlowLayout());
```

```
add(splitPane);
setVisible(true);
}
}

class JSplitPaneEx
{
public static void main(String[]ar)
{
Abc a1=new Abc();
}
}
```

Output:



Q11. How to denote the user about the software loading process? Which component is facilitating the same? Explain with code specification.

- This is the class which creates the progress bar using its constructor JProgressBar () to show the status of our process completion.
- The constructor JProgressBar() takes two argument as parameter in which, first is the initial value of the progress bar which is shown in the starting and another argument is the counter value by which the value of the progress bar is incremented.

setStringPainted(boolean):
This is the method of the JProgressBar class which shows the complete process in percent on the progress bar. It takes a Boolean value as a parameter. If we pass the true then the value will be seen on the progress bar otherwise not seen.

setValue():
This is the method of the JProgressBar class which sets the value to the progress bar.

Timer ():
This constructor takes two arguments as parameter first is the interval (in milliseconds) of the timer and second one is the listener object. Time is started using the start () method of the Timer class.

Example of JProgressBar with source code

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

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

```
JFrame f = new JFrame("JProgressBar Sample");
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JProgressBar progressBar = new JProgressBar();
progressBar.setValue(25);
progressBar.setStringPainted(true);
Border border = BorderFactory.createTitledBorder("Reading...");
progressBar.setBorder(border);
f.add(progressBar, BorderLayout.NORTH);
f.setSize(300, 100);
f.setVisible(true);
}
}
```