GUI and Java Swing

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 as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

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

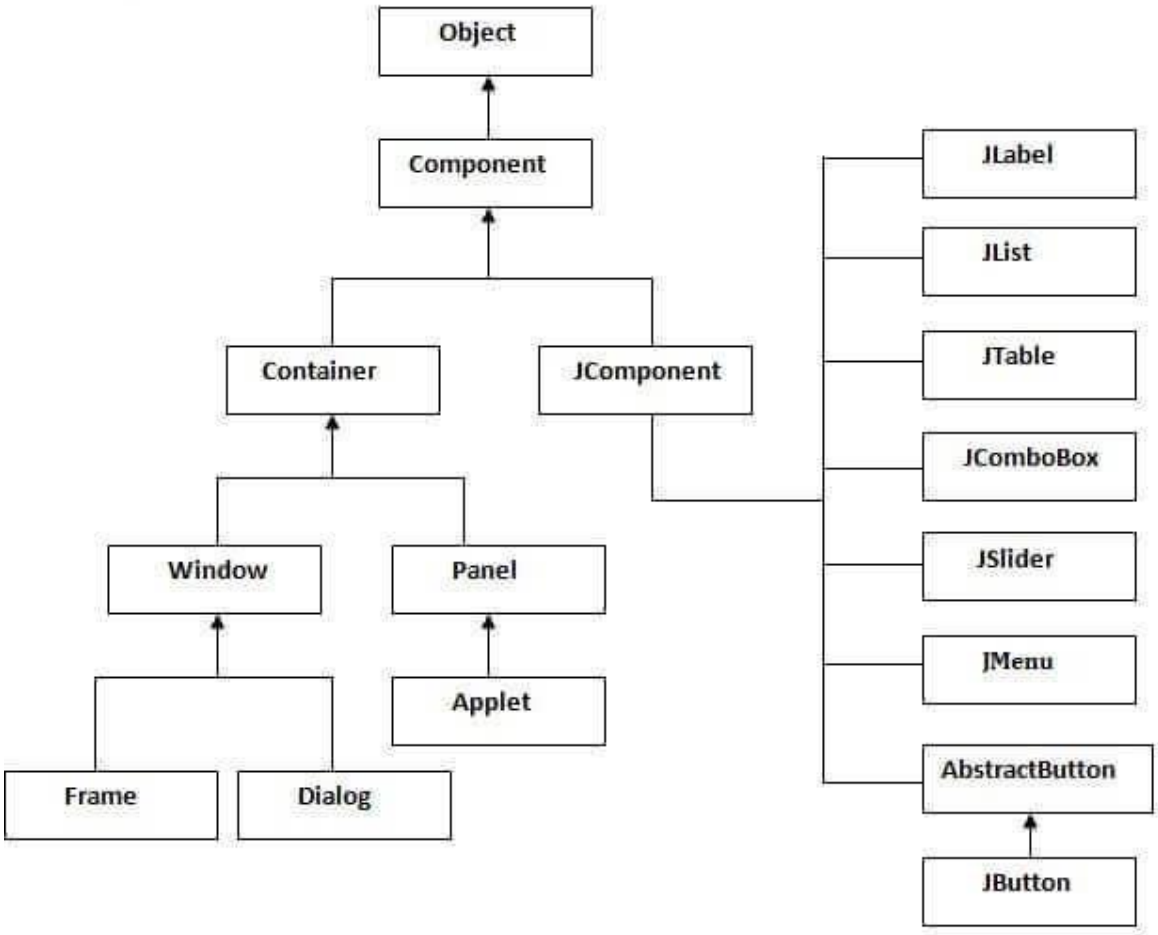
No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT 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**.

What is JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.

public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Java Swing Examples

There are two ways to create a frame:

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

We can write the code of swing inside the main(), constructor or any other method.

Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

File: FirstSwingExample.java

```
1. import javax.swing.*;
2. public class FirstSwingExample {
3. public static void main(String[] args) {
4. JFrame f=new JFrame();//creating instance of JFrame
5.
6. JButton b=new JButton("click");//creating instance of JButton
7. b.setBounds(130,100,100, 40);//x axis, y axis, width, height
8.
9. f.add(b);//adding button in JFrame
10.
11. f.setSize(400,500);//400 width and 500 height
12. f.setLayout(null);//using no layout managers
13. f.setVisible(true);//making the frame visible
```

```
14. }
15. }
```

Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

File: Simple.java

```
1.  import javax.swing.*;
2.  public class Simple {
3.  JFrame f;
4.  Simple(){
5.  f=new JFrame();//creating instance of JFrame
6.
7.  JButton b=new JButton("click");//creating instance of JButton
8.  b.setBounds(130,100,100, 40);
9.
10. f.add(b);//adding button in JFrame
11.
12. f.setSize(400,500);//400 width and 500 height
13. f.setLayout(null);//using no layout managers
14. f.setVisible(true);//making the frame visible
15. }
16.
17. public static void main(String[] args) {
18. new Simple();
19. }
```

```
20. }
```

Next →← Prev

Java JComponent

The JComponent class is the base class of all Swing components except top-level containers. Swing components whose names begin with "J" are descendants of the JComponent class. For example, JButton, JScrollPane, JPanel, JTable etc. But, JFrame and JDialog don't inherit JComponent class because they are the child of top-level containers.

The JComponent class extends the Container class which itself extends Component. The Container class has support for adding components to the container.

Fields

Modifier and Type	Field	Description
protected AccessibleContext	accessibleContext	The AccessibleContext associated with this JComponent.
protectedEventListenerList	listenerList	A list of event listeners for this component.
static String	TOOL_TIP_TEXT_KEY	The comment to display when the cursor is over the component, also known as a "value tip", "flyover help", or "flyover label"
protected ComponentUI	ui	The look and feel delegate for this component.

static int	UNDEFINED_CONDITION	It is a constant used by some of the APIs to mean that no condition is defined.
static int	WHEN_ANCESTOR_OF_FOCUSED_COMPONENT	It is a constant used for registerKeyboardAction that means that the command should be invoked when the receiving component is an ancestor of the focused component or is itself the focused component.
static int	WHEN_FOCUSED	It is a constant used for registerKeyboardAction that means that the command should be invoked when the component has the focus.
static int	WHEN_IN_FOCUSED_WINDOW	Constant used for registerKeyboardAction that means that the command should be invoked when the receiving component is in the window that has the focus or is itself the focused component.

Constructor

Constructor	Description
JComponent()	Default JComponent constructor.

Useful Methods

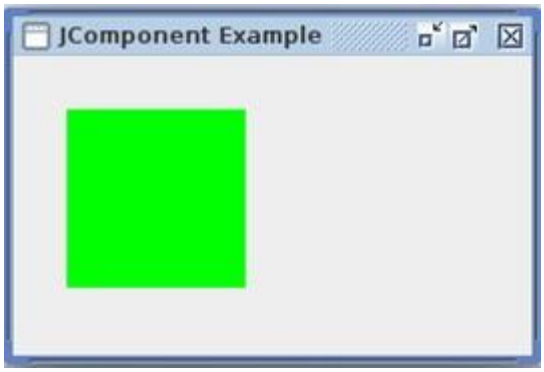
Modifier and Type	Method	Description
void	setActionMap(ActionMap am)	It sets the ActionMap to am.
void	setBackground(Color bg)	It sets the background color of this component.
void	setFont(Font font)	It sets the font for this component.
void	setMaximumSize(Dimension maximumSize)	It sets the maximum size of this component to a constant value.
void	setMinimumSize(Dimension minimumSize)	It sets the minimum size of this component to a constant value.
protected void	setUI(ComponentUI newUI)	It sets the look and feel delegate for this component.
void	setVisible(boolean aFlag)	It makes the component visible or invisible.
void	setForeground(Color fg)	It sets the foreground color of this component.
String	getToolTipText(MouseEvent event)	It returns the string to be used as the tooltip for event.
Container	getTopLevelAncestor()	It returns the top-level ancestor of this component (either the containing Window or Applet), or null if this component has not been added to any container.

TransferHandler	getTransferHandler()	It gets the transferHandler property.

Java JComponent Example

```
1. import java.awt.Color;
2. import java.awt.Graphics;
3. import javax.swing.JComponent;
4. import javax.swing.JFrame;
5. class MyJComponent extends JComponent {
6.       public void paint(Graphics g) {
7.       g.setColor(Color.green); 8.
          g.fillRect(30, 30, 100, 100);
9.       }
10. }
11. public class JComponentExample {
12.       public static void main(String[] arguments) {
13.        MyJComponent com = new MyJComponent();
14.        // create a basic JFrame
15.        JFrame.setDefaultLookAndFeelDecorated(true);
16.        JFrame frame = new JFrame("JComponent Example");
17.        frame.setSize(300,200);
18.        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.        // add the JComponent to main frame
20.        frame.add(com);
21.        frame.setVisible(true);
22.       }
23. }
```

Output:



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 such as Frame, Dialog and Panel.

Java LayoutManagers

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. **java.awt.BorderLayout**
2. **java.awt.FlowLayout**
3. **java.awt.GridLayout**
4. **java.awt.CardLayout**
5. **java.awt.GridBagLayout**
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

Java BorderLayout

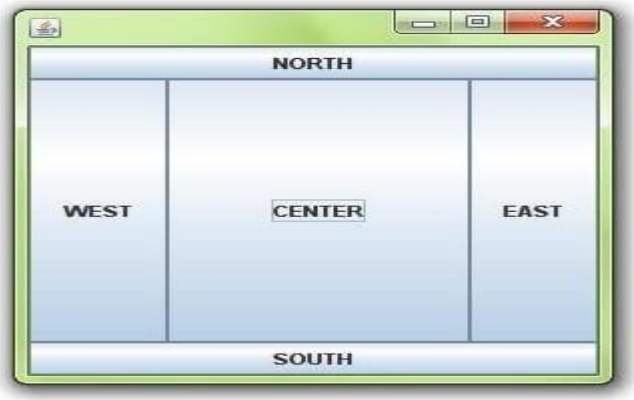
The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

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

Constructors of BorderLayout class:

- **BorderLayout():** creates a border layout but with no gaps between the components.
- **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

Example of BorderLayout class:




```
1. import java.awt.*;
2. import javax.swing.*;
3.
4. public class Border {
5. JFrame f;
```

```
6.  Border(){
7.      f=new JFrame();
8.
9.      JButton b1=new JButton("NORTH");;
10.     JButton b2=new JButton("SOUTH");;
11.     JButton b3=new JButton("EAST");;
12.     JButton b4=new JButton("WEST");;
13.     JButton b5=new JButton("CENTER");;
14.
15.     f.add(b1,BorderLayout.NORTH);
16.     f.add(b2,BorderLayout.SOUTH);
17.     f.add(b3,BorderLayout.EAST);
18.     f.add(b4,BorderLayout.WEST);
19.     f.add(b5,BorderLayout.CENTER);
20.
21.     f.setSize(300,300);
22.     f.setVisible(true);
23. }
24. public static void main(String[] args) {
25.     new Border();
26. }
27. }
```

Java GridLayout

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

Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.

3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

Example of GridLayout class


```
1.  import java.awt.*;
2.  import javax.swing.*;
3.
4.  public class MyGridLayout{
5.  JFrame f;
6.  MyGridLayout(){
7.      f=new JFrame();
8.
9.      JButton b1=new JButton("1");
10.     JButton b2=new JButton("2");
11.     JButton b3=new JButton("3");
12.     JButton b4=new JButton("4");
13.     JButton b5=new JButton("5");
14.         JButton b6=new JButton("6");
15.         JButton b7=new JButton("7");
16.     JButton b8=new JButton("8");
17.         JButton b9=new JButton("9");
18.
19.     f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
20.     f.add(b6);f.add(b7);f.add(b8);f.add(b9);
```

```
21.21.
22.     f.setLayout(new GridLayout(3,3));
23.     //setting grid layout of 3 rows and 3 columns
24.
25.     f.setSize(300,300);
26.     f.setVisible(true);
27. }
28. public static void main(String[] args) {
29.     new MyGridLayout();
30. }
31. }
```

Java 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

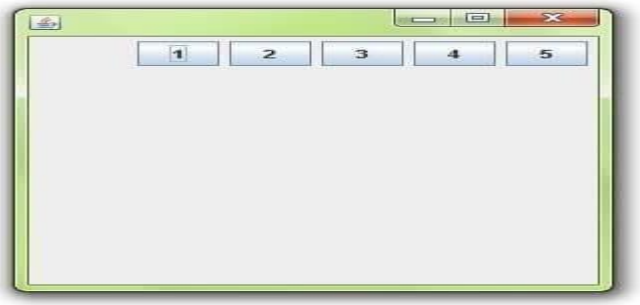
1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Constructors of FlowLayout class

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.

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

Example of FlowLayout class




```
1. import java.awt.*;
2. import javax.swing.*;
3.
4. public class MyFlowLayout{
5. JFrame f;
6. MyFlowLayout(){
7.     f=new JFrame();
8.
9.     JButton b1=new JButton("1");
10.    JButton b2=new JButton("2");
11.    JButton b3=new JButton("3");
12.    JButton b4=new JButton("4");
13.    JButton b5=new JButton("5");
14.
15.    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
16.
17.    f.setLayout(new FlowLayout(FlowLayout.RIGHT));
18.    //setting flow layout of right alignment
19.
20.    f.setSize(300,300);
21.    f.setVisible(true);
22. }
23. public static void main(String[] args) {
24.    new MyFlowLayout();
```

```
25. }
26. }
```

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.

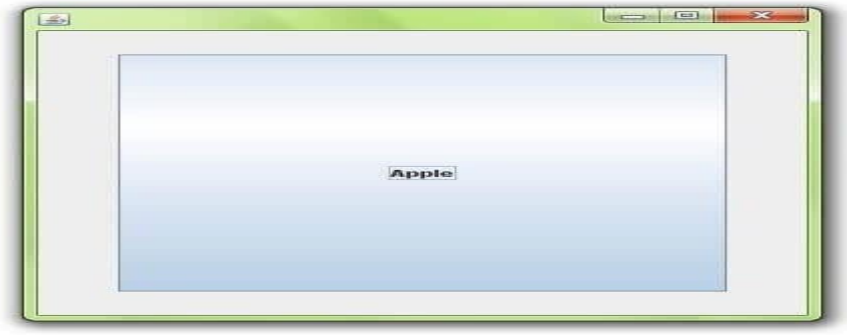
Constructors of CardLayout class

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.

Example of CardLayout class



```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.
4. import javax.swing.*;
5.
6.  public class CardLayoutExample extends JFrame implements ActionListener{
7.  CardLayout card;
8.  JButton b1,b2,b3;
9.  Container c;
10.     CardLayoutExample(){
11.11.
12.         c=getContentPane();
13.         card=new CardLayout(40,30);
14. //create CardLayout object with 40 hor space and 30 ver space
15.         c.setLayout(card);
16.16.
17.         b1=new JButton("Apple");
18.         b2=new JButton("Boy");
19.         b3=new JButton("Cat");
20.         b1.addActionListener(this);
21.         b2.addActionListener(this);
22.         b3.addActionListener(this);
23.23.
24.         c.add("a",b1);c.add("b",b2);c.add("c",b3);
25.
26.     }
27.     public void actionPerformed(ActionEvent e) {
28.     card.next(c);
29.     }
30.
31.     public static void main(String[] args) {
32.         CardLayoutExample cl=new CardLayoutExample();
33.         cl.setSize(400,400);
34.         cl.setVisible(true);
35.         cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
36.     }
```

```
37. }
```

Java 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.

Fields

Modifier and Type	Field	Description
double[]	columnWeights	It is used to hold the overrides to the column weights.
int[]	columnWidths	It is used to hold the overrides to the column minimum width.
protected Hashtable<Component,GridBagConstraints>	comptable	It is used to maintains the association between a component and its gridbag constraints.
protected GridBagConstraints	defaultConstraints	It is used to hold a gridbag constraints instance containing the default values.

protected GridBagLayoutInfo	layoutInfo	It is used to hold the layout information for the gridbag.
protected static int	MAXGRIDSIZE	No longer in use just for backward compatibility
protected static int	MINSIZE	It is smallest grid that can be laid out by the grid bag layout.
protected static int	PREFERREDSIZE	It is preferred grid size that can be laid out by the grid bag layout.
int[]	rowHeights	It is used to hold the overrides to the row minimum heights.
double[]	rowWeights	It is used to hold the overrides to the row weights.

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.
float	getLayoutAlignmentY(Container parent)	It returns the alignment along the y axis.
int[][]	getLayoutDimensions()	It determines column widths and row heights for the layout grid.
protected GridBagLayoutInfo	getLayoutInfo(Container parent, int sizeflag)	This method is obsolete and supplied for backwards compatibility.
protected GridBagLayoutInfo	GetLayoutInfo(Container parent, int sizeflag)	This method is obsolete and supplied for backwards compatibility.

Point	getLayoutOrigin()	It determines the origin of the layout area, in the graphics coordinate space of the target container.
double[][]	getLayoutWeights()	It determines the weights of the layout grid's columns and rows.
protected Dimension	getMinSize(Container parent, GridBagLayoutInfo info)	It figures out the minimum size of the master based on the information from getLayoutInfo.
protected Dimension	GetMinSize(Container parent, GridBagLayoutInfo info)	This method is obsolete and supplied for backwards compatibility only

Example

```
1. import java.awt.Button;
2. import java.awt.GridBagConstraints;
3. import java.awt.GridBagLayout;
4.
5. import javax.swing.*;
6. public class GridBagLayoutExample extends JFrame{
7.     public static void main(String[] args) {
8.           GridBagLayoutExample a = new GridBagLayoutExample();
9.        }
10.      public GridBagLayoutExample() {
11.   GridBagLayoutgrid = new GridBagLayout();
12.          GridBagConstraints gbc = new GridBagConstraints();
13.           setLayout(grid);
14.          setTitle("GridBag Layout Example");
15.           GridBagLayout layout = new GridBagLayout();
16.   this.setLayout(layout);
17.   gbc.fill = GridBagConstraints.HORIZONTAL;
```

```
18.     gbc.gridx = 0;
19.     gbc.gridy = 0;
20.     this.add(new Button("Button One"), gbc);
21.     gbc.gridx = 1;
22.     gbc.gridy = 0;
23.     this.add(new Button("Button two"), gbc);
24.     gbc.fill = GridBagConstraints.HORIZONTAL;
25.     gbc.ipady = 20;
26.     gbc.gridx = 0;
27.     gbc.gridy = 1;
28.     this.add(new Button("Button Three"), gbc);
29.     gbc.gridx = 1;
30.     gbc.gridy = 1;
31.     this.add(new Button("Button Four"), gbc);
32.     gbc.gridx = 0;
33.     gbc.gridy = 2;
34.     gbc.fill = GridBagConstraints.HORIZONTAL;
35.     gbc.gridwidth = 2;
36.     this.add(new Button("Button Five"), gbc);
37.             setSize(300, 300);
38.             setPreferredSize(getSize());
39.             setVisible(true);
40.             setDefaultCloseOperation(EXIT_ON_CLOSE);
41.41.
42.         }
43.
44. }
```

Java 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 with ActionListener

```
1. import java.awt.event.*;
2. import javax.swing.*;
3. public class ButtonExample {
```

```
4.  public static void main(String[] args) {
5.      JFrame f=new JFrame("Button Example");
6.      final JTextField tf=new JTextField();
7.      tf.setBounds(50,50, 150,20);
8.      JButton b=new JButton("Click Here");
9.      b.setBounds(50,100,95,30);
10.     b.addActionListener(new ActionListener(){
11. public void actionPerformed(ActionEvent e){
12.             tf.setText("Welcome to CMR.");
13.         }
14.     });
15.     f.add(b);f.add(tf);
16.     f.setSize(400,400);
17.     f.setLayout(null);
18.     f.setVisible(true);
19. }
20. }
```

Java JLabel

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

JLabel class declaration

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

```
1. public class JLabel extends JComponent implements SwingConstants, Accessible
```

Commonly used Constructors:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.

JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

Commonly used Methods:

Methods	Description
String getText()	t returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X axis.
Icon getIcon()	It returns the graphic image that the label displays.
int getHorizontalAlignment()	It returns the alignment of the label's contents along the X axis.

Java JLabel Example with ActionListener

```
1. import javax.swing.*;
2. import java.awt.*;
3. import java.awt.event.*;
4. public class LabelExample extends Frame implements ActionListener{
5.     JTextField tf; JLabel l; JButton b;
6.     LabelExample(){
7.         tf=new JTextField();
8.         tf.setBounds(50,50, 150,20);
9.         l=new JLabel();
10.        l.setBounds(50,100, 150,20);
11.        b=new JButton("Find IP");
```

```
12.       b.setBounds(50,150,95,30);
13.       b.addActionListener(this);
14.       add(b);add(tf);add(l);
15.       setSize(400,400);
16.       setLayout(null);
17.       setVisible(true);
18.   }
19.   public void actionPerformed(ActionEvent e) {
20.       try{
21.       String host=tf.getText();
22.       String ip=java.net.InetAddress.getByName(host).getHostAddress();
23.       l.setText("IP of "+host+" is: "+ip);
24.       }catch(Exception ex){System.out.println(ex);}
25.   }
26.   public static void main(String[] args) {
27.   new LabelExample();
28.   } }
```

Java JTextField

The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

JTextField class declaration

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

1. **public class** JTextField **extends** JTextComponent **implements** SwingConstants

Commonly used Constructors:

Constructor	Description
JTextField()	Creates a new TextField

JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

Commonly used Methods:

Methods	Description
void addActionListener(ActionListener l)	It is used to add the specified action listener to receive action events from this textfield.
Action getAction()	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
void setFont(Font f)	It is used to set the current font.
void removeActionListener(ActionListener l)	It is used to remove the specified action listener so that it no longer receives action events from this textfield.

Java JTextField Example with ActionListener

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class TextFieldExample implements ActionListener{
4.     JTextField tf1,tf2,tf3;
5.     JButton b1,b2;
6.     TextFieldExample(){
7.         JFrame f= new JFrame();
8.         tf1=new JTextField();
9.         tf1.setBounds(50,50,150,20);
10.        tf2=new JTextField();
11.        tf2.setBounds(50,100,150,20);
```

```
12.         tf3=new JTextField();
13.         tf3.setBounds(50,150,150,20);
14.         tf3.setEditable(false);
15.         b1=new JButton("+");
16.         b1.setBounds(50,200,50,50);
17.         b2=new JButton("-");
18.         b2.setBounds(120,200,50,50);
19.         b1.addActionListener(this);
20.         b2.addActionListener(this);
21.         f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);f.add(b2);
22.         f.setSize(300,300);
23.         f.setLayout(null);
24.         f.setVisible(true);
25.     }
26.     public void actionPerformed(ActionEvent e) {
27.         String s1=tf1.getText();
28.         String s2=tf2.getText();
29.         int a=Integer.parseInt(s1);
30.         int b=Integer.parseInt(s2);
31.         int c=0;
32.         if(e.getSource()==b1){
33.             c=a+b;
34.         }else if(e.getSource()==b2){
35.             c=a-b;
36.         }
37.         String result=String.valueOf(c);
38.         tf3.setText(result);
39.     }
40. public static void main(String[] args) {
41.     new TextFieldExample();
42. } }
```

Java JCheckBox

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.

1. **public class** JCheckBox **extends** JToggleButton **implements** Accessible

Commonly used Constructors:

Constructor	Description
JCheckBox()	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:

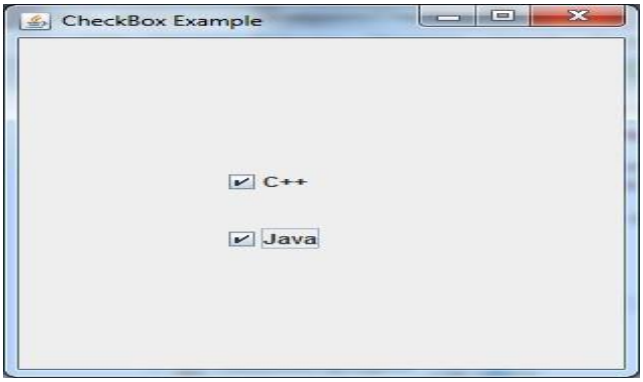
Methods	Description

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

Java JCheckBox Example

```
1.  import javax.swing.*;
2.  public class CheckBoxExample
3.  {
4.       CheckBoxExample(){
5.          JFrame f= new JFrame("CheckBox Example");
6.         JCheckBox checkBox1 = new JCheckBox("C++");
7.          checkBox1.setBounds(100,100, 50,50);
8.         JCheckBox checkBox2 = new JCheckBox("Java", true);
9.          checkBox2.setBounds(100,150, 50,50);
10.        f.add(checkBox1);
11.         f.add(checkBox2);
12.        f.setSize(400,400);
13.         f.setLayout(null);
14.        f.setVisible(true);
15.     }
16. public static void main(String args[])
17.    {
18.    new CheckBoxExample();
19.    }}
```

Output:



Java 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.

JRadioButton class declaration

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

1. **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.

Java JRadioButton Example

```
1. import javax.swing.*;
2. public class RadioButtonExample {
3. JFrame f;
4. RadioButtonExample(){
5. f=new JFrame();
6. JRadioButton r1=new JRadioButton("A) Male");
7. JRadioButton r2=new JRadioButton("B) Female");
8. r1.setBounds(75,50,100,30);
9. r2.setBounds(75,100,100,30);
10. ButtonGroup bg=new ButtonGroup();
11. bg.add(r1);bg.add(r2);
12. f.add(r1);f.add(r2);
13. f.setSize(300,300);
14. f.setLayout(null);
15. f.setVisible(true);
16. }
17. public static void main(String[] args) {
```

```
18.     new RadioButtonExample();
19. }
20. }
```

Java JComboBox

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits JComponent class.

JComboBox class declaration

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

```
1. public class JComboBox extends JComponent implements ItemSelectable, ListDataL
   istener, ActionListener, Accessible
```

Commonly used Constructors:

Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified array.
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified Vector.

Commonly used Methods:

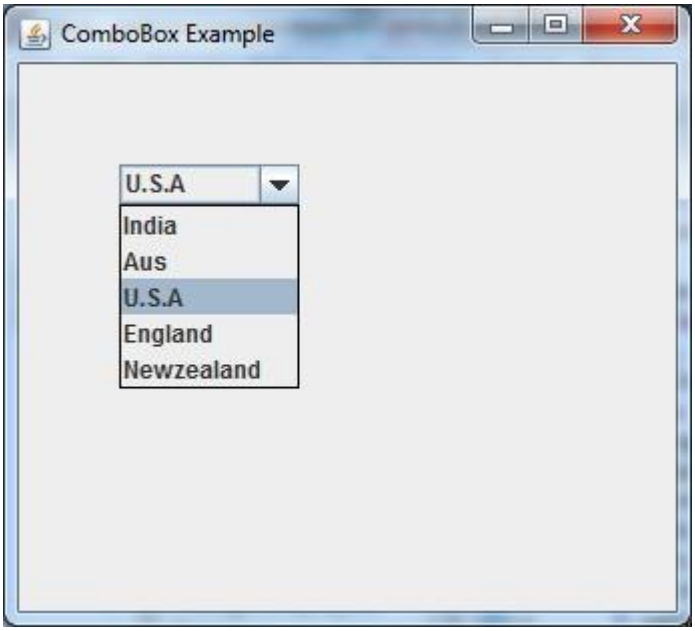
Methods	Description

void addItem(Object anObject)	It is used to add an item to the item list.
void removeItem(Object anObject)	It is used to delete an item to the item list.
void removeAllItems()	It is used to remove all the items from the list.
void setEditable(boolean b)	It is used to determine whether the JComboBox is editable.
void addActionListener(ActionListener a)	It is used to add the ActionListener.
void addItemListener(ItemListener i)	It is used to add the ItemListener.

Java JComboBox Example

```
1. import javax.swing.*;
2. public class ComboBoxExample {
3. JFrame f;
4. ComboBoxExample(){
5.     f=new JFrame("ComboBox Example");
6.     String country[]={"India","Aus","U.S.A","England","Newzealand"};
7.     JComboBox cb=new JComboBox(country);
8.     cb.setBounds(50, 50,90,20);
9.     f.add(cb);
10.    f.setLayout(null);
11.    f.setSize(400,500);
12.    f.setVisible(true);
13. }
14. public static void main(String[] args) {
15.    new ComboBoxExample();
16. }
17. }
```

Output:



Java JTabbedPane

The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.

JTabbedPane class declaration

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

1. **public class** JTabbedPane **extends** JComponent **implements** Serializable, Accessible, SwingConstants

Commonly used Constructors:

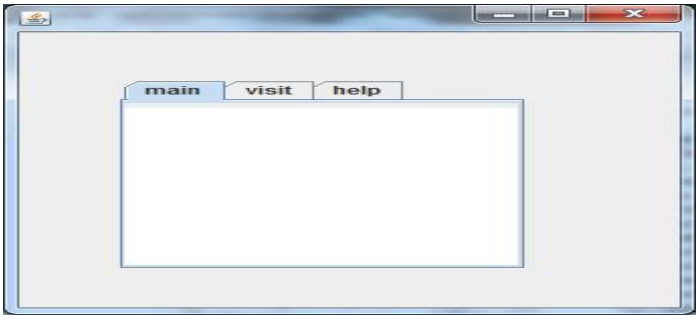
Constructor	Description
JTabbedPane()	Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top.
JTabbedPane(int tabPlacement)	Creates an empty TabbedPane with a specified tab placement.

JTabbedPane(int tabPlacement, int tabLayoutPolicy)	Creates an empty TabbedPane with a specified tab placement and tab layout policy.

Java JTabbedPane Example

```
1. import javax.swing.*;
2. public class TabbedPaneExample {
3. JFrame f;
4. TabbedPaneExample(){
5.     f=new JFrame();
6.     JTextArea ta=new JTextArea(200,200);
7.     JPanel p1=new JPanel();
8.     p1.add(ta);
9.     JPanel p2=new JPanel();
10.    JPanel p3=new JPanel();
11.    JTabbedPane tp=new JTabbedPane();
12.    tp.setBounds(50,50,200,200);
13.    tp.add("main",p1);
14.    tp.add("visit",p2);
15.    tp.add("help",p3);
16.    f.add(tp);
17.    f.setSize(400,400);
18.    f.setLayout(null);
19.    f.setVisible(true);
20. }
21. public static void main(String[] args) {
22.    new TabbedPaneExample();
23. }}
```

Output:



Java JScrollPane

A JscrollPane is used to make scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.

Constructors

Constructor	Purpose
JScrollPane()	It creates a scroll pane. The Component parameter, when present, sets the scroll pane's client. The two int parameters, when present, set the vertical and horizontal scroll bar policies (respectively).
JScrollPane(Component)	
JScrollPane(int, int)	
JScrollPane(Component, int, int)	

Useful Methods

Modifier	Method	Description
void	setColumnHeaderView(Component)	It sets the column header for the scroll pane.
void	setRowHeaderView(Component)	It sets the row header for the scroll pane.
void	setCorner(String, Component)	

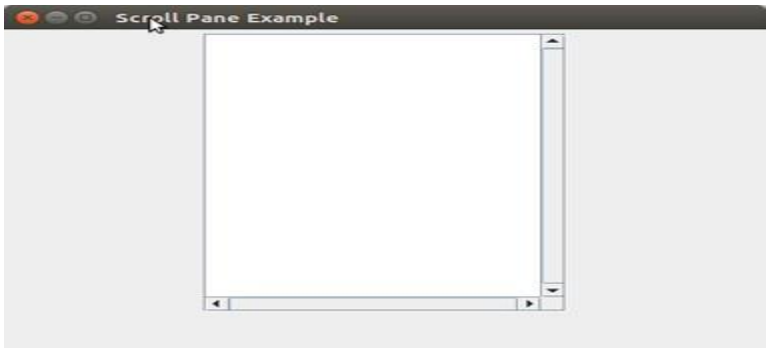
Component	getCorner(String)	It sets or gets the specified corner. The int parameter specifies which corner and must be one of the following constants defined in ScrollPaneConstants: UPPER_LEFT_CORNER, UPPER_RIGHT_CORNER, LOWER_LEFT_CORNER, LOWER_RIGHT_CORNER, LOWER_LEADING_CORNER, LOWER_TRAILING_CORNER, UPPER_LEADING_CORNER, UPPER_TRAILING_CORNER.
void	setViewportView(Component)	Set the scroll pane's client.

JScrollPane Example

```
1.  import java.awt.FlowLayout;
2.  import javax.swing.JFrame;
3.  import javax.swing.JScrollPane;
4.  import javax.swing.JtextArea;
5.
6.  public class JScrollPaneExample {
7.      private static final long serialVersionUID = 1L;
8.
9.      private static void createAndShowGUI() {
10.
11.         // Create and set up the window.
12.         final JFrame frame = new JFrame("Scroll Pane Example");
13.
14.         // Display the window.
15.         frame.setSize(500, 500);
16.         frame.setVisible(true);
17.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18.
19.         // set flow layout for the frame
20.         frame.getContentPane().setLayout(new FlowLayout());
```

```
21.21.
22.         JTextArea textArea = new JTextArea(20, 20);
23.         JScrollPane scrollableTextArea = new JScrollPane(textArea);
24.
25.         scrollableTextArea.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROL
    LBAR_ALWAYS);
26.         scrollableTextArea.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_A
    LWAYS);
27.27.
28.         frame.getContentPane().add(scrollableTextArea);
29.     }
30.     public static void main(String[] args) {
31.
32.
33.         javax.swing.SwingUtilities.invokeLater(new Runnable() {
34.
35.             public void run() {
36.                 createAndShowGUI();
37.             }
38.         });
39.     }
40. }
```

Output:



Java JToggleButton

JToggleButton is used to create toggle button, it is two-states button to switch on or off.

Nested Classes

Modifier and Type	Class	Description
protected class	JToggleButton.AccessibleJToggleButton	This class implements accessibility support for the JToggleButton class.
static class	JToggleButton.ToggleButtonModel	The ToggleButton model

Constructors

Constructor	Description
JToggleButton()	It creates an initially unselected toggle button without setting the text or image.
JToggleButton(Action a)	It creates a toggle button where properties are taken from the Action supplied.
JToggleButton(Icon icon)	It creates an initially unselected toggle button with the specified image but no text.
JToggleButton(Icon icon, boolean selected)	It creates a toggle button with the specified image and selection state, but no text.
JToggleButton(String text)	It creates an unselected toggle button with the specified text.
JToggleButton(String text, boolean selected)	It creates a toggle button with the specified text and selection state.
JToggleButton(String text, Icon icon)	It creates a toggle button that has the specified text and image, and that is initially unselected.
JToggleButton(String text, Icon icon, boolean selected)	It creates a toggle button with the specified text, image, and selection state.

Methods

Modifier and Type	Method	Description
AccessibleContext	getAccessibleContext()	It gets the AccessibleContext associated with this JToggleButton.
String	getUIClassID()	It returns a string that specifies the name of the l&f class that renders this component.
protected String	paramString()	It returns a string representation of this JToggleButton.
void	updateUI()	It resets the UI property to a value from the current look and feel.

JToggleButton Example

```
1. import java.awt.FlowLayout;
2. import java.awt.event.ItemEvent;
3. import java.awt.event.ItemListener;
4. import javax.swing.JFrame;
5. import javax.swing.JToggleButton;
6.
7. public class JToggleButtonExample extends JFrame implements ItemListener {
8.     public static void main(String[] args) {
9.         new JToggleButtonExample();
10.    }
11.    private JToggleButton button;
12.    JToggleButtonExample() {
13.        setTitle("JToggleButton with ItemListener Example");
14.        setLayout(new FlowLayout());
15.        setJToggleButton();
16.        setAction();
17.        setSize(200, 200);
18.        setVisible(true);
19.        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20.    }
```

```
21.    private void setJToggleButton() {
22.        button = new JToggleButton("ON");
23.        add(button);
24.    }
25.    private void setAction() {
26.        button.addItemListener(this);
27.    }
28.    public void itemStateChanged(ItemEvent eve) {
29.        if (button.isSelected())
30.            button.setText("OFF");
31.        else
32.            button.setText("ON");
33.    }
34. }
```

Output

Java 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

```
1. public class JMenuBar extends JComponent implements MenuElement, Accessible
```

JMenu class declaration

```
1. public class JMenu extends JMenuItem implements MenuElement, Accessible
```

JMenuItem class declaration

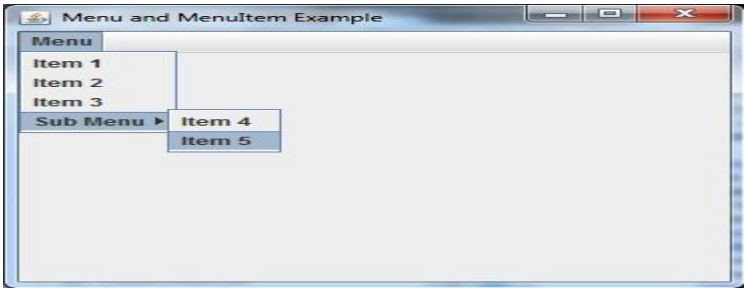
```
1. public class JMenuItem extends AbstractButton implements Accessible, MenuEleme
   nt
```

Java JMenuItem and JMenu Example

```
1.  import javax.swing.*;
2.  class MenuExample
3. {
4.          JMenu menu, submenu;
5.          JMenuItem i1, i2, i3, i4, i5;
6.          MenuExample(){
7.          JFrame f= new JFrame("Menu and MenuItem Example");
8.          JMenuBar mb=new JMenuBar();
9.          menu=new JMenu("Menu");
10.         submenu=new JMenu("Sub Menu");
11.         i1=new JMenuItem("Item 1");
12.         i2=new JMenuItem("Item 2");
13.         i3=new JMenuItem("Item 3");
14.         i4=new JMenuItem("Item 4");
15.         i5=new JMenuItem("Item 5");
16.         menu.add(i1); menu.add(i2); menu.add(i3);
17.         submenu.add(i4); submenu.add(i5);
18.         menu.add(submenu);
19.         mb.add(menu);
20.         f.setJMenuBar(mb);
21.         f.setSize(400,400);
22.         f.setLayout(null);
23.         f.setVisible(true);
24. }
```

```
25. public static void main(String args[])
26. {
27. new MenuExample();
28. }}
```

Output:



Example of creating Edit menu for Notepad:

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class MenuExample implements ActionListener{
4. JFrame f;
5. JMenuBar mb;
6. JMenu file,edit,help;
7. JMenuItem cut,copy,paste,selectAll;
8. JTextArea ta;
9. MenuExample(){
10. f=new JFrame();
11. cut=new JMenuItem("cut");
12. copy=new JMenuItem("copy");
13. paste=new JMenuItem("paste");
14. selectAll=new JMenuItem("selectAll");
15. cut.addActionListener(this);
16. copy.addActionListener(this);
17. paste.addActionListener(this);
18. selectAll.addActionListener(this);
19. mb=new JMenuBar();
20. file=new JMenu("File");
21. edit=new JMenu("Edit");
```

```
22. help=new JMenu("Help");
23. edit.add(cut);edit.add(copy);edit.add(paste);edit.add(selectAll);
24. mb.add(file);mb.add(edit);mb.add(help);
25. ta=new JTextArea();
26. ta.setBounds(5,5,360,320);
27. f.add(mb);f.add(ta);
28. f.setJMenuBar(mb);
29. f.setLayout(null);
30. f.setSize(400,400);
31. f.setVisible(true);
32. }
33. public void actionPerformed(ActionEvent e) {
34. if(e.getSource()==cut)
35. ta.cut();
36. if(e.getSource()==paste)
37. ta.paste();
38. if(e.getSource()==copy)
39. ta.copy();
40. if(e.getSource()==selectAll)
41. ta.selectAll();
42. }
43. public static void main(String[] args) {
44.     new MenuExample();
45. }
46. }
```

Output:

Java JDialog

The JDialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Dialog class.

Unlike JFrame, it doesn't have maximize and minimize buttons.

JDialog class declaration

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

1. **public class** JDialog **extends** Dialog **implements** WindowConstants, Accessible, Root PaneContainer

Commonly used Constructors:

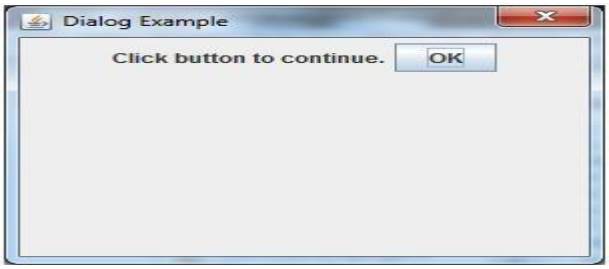
Constructor	Description
JDialog()	It is used to create a modeless dialog without a title and without a specified Frame owner.
JDialog(Frame owner)	It is used to create a modeless dialog with specified Frame as its owner and an empty title.
JDialog(Frame owner, String title, boolean modal)	It is used to create a dialog with the specified title, owner Frame and modality.

Java JDialog Example

```
1. import javax.swing.*;
2. import java.awt.*;
3. import java.awt.event.*;
4. public class DialogExample {
5.     private static JDialog d;
6.     DialogExample() {
7.         JFrame f= new JFrame();
8.         d = new JDialog(f , "Dialog Example", true);
9.         d.setLayout( new FlowLayout() );
```

```
10.       JButton b = new JButton ("OK");
11.       b.addActionListener ( new ActionListener()
12.       {
13.           public void actionPerformed( ActionEvent e )
14.           {
15.               DialogExample.d.setVisible(false);
16.           }
17.       });
18.       d.add( new JLabel ("Click button to continue."));
19.       d.add(b);
20.       d.setSize(300,300);
21.       d.setVisible(true);
22.   }
23.   public static void main(String args[])
24.   {
25.       new DialogExample();
26.   }
27. }
```

Output:



Delegation Event Model in Java

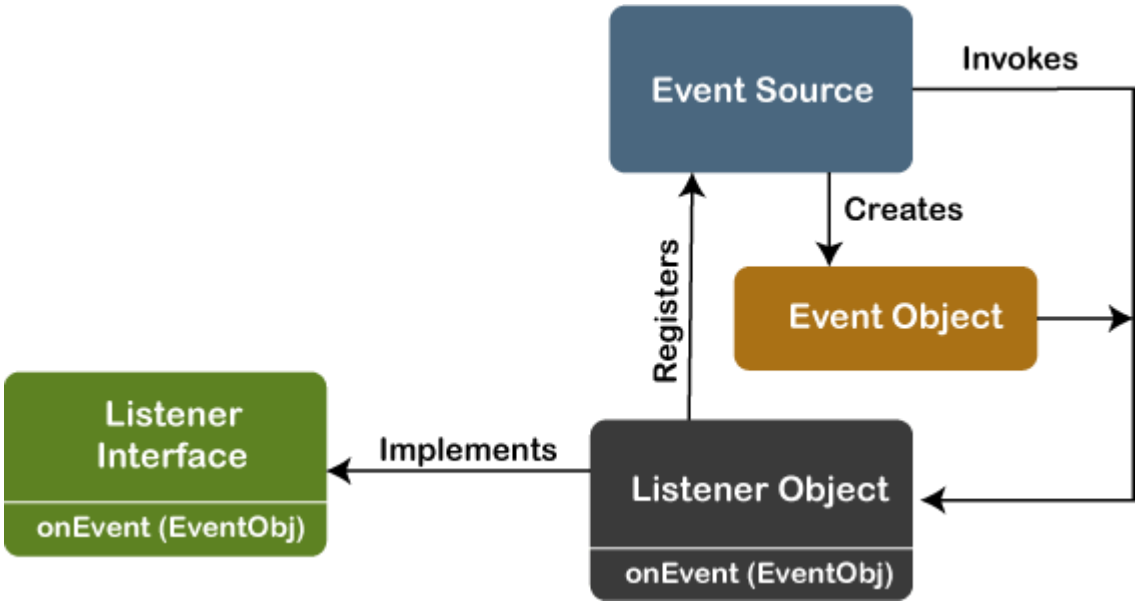
The Delegation Event model is defined to handle events in GUI programming languages. The GUI stands for Graphical User Interface, where a user graphically/visually interacts with the system.

The GUI programming is inherently event-driven; whenever a user initiates an activity such as a mouse activity, clicks, scrolling, etc., each is known as an event that is mapped to a code to respond to functionality to the user. This is known as event handling.

In this section, we will discuss event processing and how to implement the delegation event model in Java. We will also discuss the different components of an Event Model.

Event Processing in Java

Java support event processing since Java 1.0. It provides support for AWT (Abstract Window Toolkit), which is an API used to develop the Desktop application. In Java 1.0, the AWT was based on inheritance. To catch and process GUI events for a program, it should hold subclass GUI components and override action() or handleEvent() methods. The below image demonstrates the event processing.



But, the modern approach for event processing is based on the Delegation Model. It defines a standard and compatible mechanism to generate and process events. In this model, a source generates an event and forwards it to one or more listeners. The listener waits until it receives an event. Once it receives the event, it is processed by the listener and returns it. The UI elements are able to delegate the processing of an event to a separate function.

The key advantage of the Delegation Event Model is that the application logic is completely separated from the interface logic.

In this model, the listener must be connected with a source to receive the event notifications. Thus, the events will only be received by the listeners who wish to receive them. So, this approach is more convenient than the inheritance-based event model (in Java 1.0).

In the older model, an event was propagated up the containment until a component was handled. This needed components to receive events that were not processed, and it took lots of time. The Delegation Event model overcame this issue.

Basically, an Event Model is based on the following three components:

- Events
- Events Sources
- Events Listeners

Events

The Events are the objects that define state change in a source. An event can be generated as a reaction of a user while interacting with GUI elements. Some of the event generation activities are moving the mouse pointer, clicking on a button, pressing the keyboard key, selecting an item from the list, and so on. We can also consider many other user operations as events.

The Events may also occur that may be not related to user interaction, such as a timer expires, counter exceeded, system failures, or a task is completed, etc. We can define events for any of the applied actions.

Event Sources

A source is an object that causes and generates an event. It generates an event when the internal state of the object is changed. The sources are allowed to generate several different types of events.

A source must register a listener to receive notifications for a specific event. Each event contains its registration method. Below is an example:

```
1. public void addTypeListener (TypeListener e1)
```

From the above syntax, the Type is the name of the event, and e1 is a reference to the event listener. For example, for a keyboard event listener, the method will be called as **addKeyListener()**. For the mouse event listener, the method will be called as **addMouseMotionListener()**. When an event is triggered using the respected source, all the events will be notified to registered listeners and receive the event object. This process is known as event multicasting. In few cases, the event notification will only be sent to listeners that register to receive them.

Some listeners allow only one listener to register. Below is an example:

```
1. public void addTypeListener(TypeListener e2) throws java.util.TooManyListenersExce
   ption
```

From the above syntax, the Type is the name of the event, and e2 is the event listener's reference. When the specified event occurs, it will be notified to the registered listener. This process is known as **unicasting** events.

A source should contain a method that unregisters a specific type of event from the listener if not needed. Below is an example of the method that will remove the event from the listener.

```
1. public void removeTypeListener(TypeListener e2?)
```

From the above syntax, the Type is an event name, and e2 is the reference of the listener. For example, to remove the keyboard listener, the **removeKeyListener()** method will be called.

The source provides the methods to add or remove listeners that generate the events. For example, the Component class contains the methods to operate on the different types of events, such as adding or removing them from the listener.

Event Listeners

An event listener is an object that is invoked when an event triggers. The listeners require two things; first, it must be registered with a source; however, it can be registered with several resources to receive notification about the events. Second, it must implement the methods to receive and process the received notifications.

The methods that deal with the events are defined in a set of interfaces. These interfaces can be found in the java.awt.event package.

For example, the **MouseMotionListener** interface provides two methods when the mouse is dragged and moved. Any object can receive and process these events if it implements the MouseMotionListener interface.

Types of Events

The events are categories into the following two categories:

The Foreground Events:

The foreground events are those events that require direct interaction of the user. These types of events are generated as a result of user interaction with the GUI component. For example, clicking on a button, mouse movement, pressing a keyboard key, selecting an option from the list, etc.

The Background Events :

The Background events are those events that result from the interaction of the end-user. For example, an Operating system interrupts system failure (Hardware or Software).

To handle these events, we need an event handling mechanism that provides control over the events and responses.

The Delegation Model

The Delegation Model is available in Java since Java 1.1. it provides a new delegation-based event model using AWT to resolve the event problems. It provides a convenient mechanism to support complex Java programs.

Design Goals

The design goals of the event delegation model are as following:

- It is easy to learn and implement
- It supports a clean separation between application and GUI code.
- It provides robust event handling program code which is less error-prone (strong compile-time checking)
- It is Flexible, can enable different types of application models for event flow and propagation.
- It enables run-time discovery of both the component-generated events as well as observable events.
- It provides support for the backward binary compatibility with the previous model.

Let's implement it with an example:

Java Program to Implement the Event Deligation Model

The below is a Java program to handle events implementing the event deligation model:

TestApp.java:

```
1. import java.awt.*;
2. import java.awt.event.*;
3. 
4. public class TestApp {
5.     public void search() {
6.         // For searching
7.         System.out.println("Searching...");
```

```
8.      }
9.      public void sort() {
10.       // for sorting
11.        System.out.println("Sorting... ");
12.     }
13.
14.     static public void main(String args[]) {
15.       TestApp app = new TestApp();
16.       GUI gui = new GUI(app);
17.     }
18. }
19.
20. class Command implements ActionListener  {
21.     static final int SEARCH = 0;
22.     static final int SORT = 1;
23.     int id;
24.     TestApp app;
25.25.
26.     public Command(int id, TestApp app) {
27.        this.id = id;
28.        this.app = app;
29.     }
30.
31.     public void actionPerformed(ActionEvent e) {
32.        switch(id) {
33.          case SEARCH:
34.           app.search();
35.            break;
36.          case SORT:
37.            app.sort();
38.           break;
39.        }
40.     }
41. }
42.
43. class GUI {
```

```
44.
45.    public GUI(TestApp app) {
46.        Frame f = new Frame();
47.        f.setLayout(new FlowLayout());
48.         Command searchCmd = new Command(Command.SEARCH, app);
49.        Command sortCmd = new Command(Command.SORT, app);
50.          Button b;
51.        f.add(b = new Button("Search"));
52.        b.addActionListener(searchCmd);
53.        f.add(b = new Button("Sort"));
54.        b.addActionListener(sortCmd);
55.          List l;
56.        f.add(l = new List());
57.        l.add("Alphabetical");
58.        l.add("Chronological");
59.        l.addActionListener(sortCmd);
60.        f.pack();
61.61.
62.        f.show();
63.    }
64.}
```

Output:

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.

Methods of MouseListener interface

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

1. **public abstract void** mouseClicked(MouseEvent e);
2. **public abstract void** mouseEntered(MouseEvent e);
3. **public abstract void** mouseExited(MouseEvent e);
4. **public abstract void** mousePressed(MouseEvent e);
5. **public abstract void** mouseReleased(MouseEvent e);

Java MouseListener Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseListenerExample extends Frame implements MouseListener{
4.     Label l;
5.     MouseListenerExample(){
6.         addMouseListener(this);
7.
8.         l=new Label();
9.         l.setBounds(20,50,100,20);
10.        add(l);
11.        setSize(300,300);
12.        setLayout(null);
13.        setVisible(true);
14.    }
15.    public void mouseClicked(MouseEvent e) {
16.        l.setText("Mouse Clicked");
17.    }
18.    public void mouseEntered(MouseEvent e) {
19.        l.setText("Mouse Entered");
20.    }
21.    public void mouseExited(MouseEvent e) {
22.        l.setText("Mouse Exited");
23.    }
24.    public void mousePressed(MouseEvent e) {
25.        l.setText("Mouse Pressed");
26.    }
27.    public void mouseReleased(MouseEvent e) {
```

```
28.         l.setText("Mouse Released");
29.     }
30. public static void main(String[] args) {
31.     new MouseListenerExample();
32. }
33. }
```

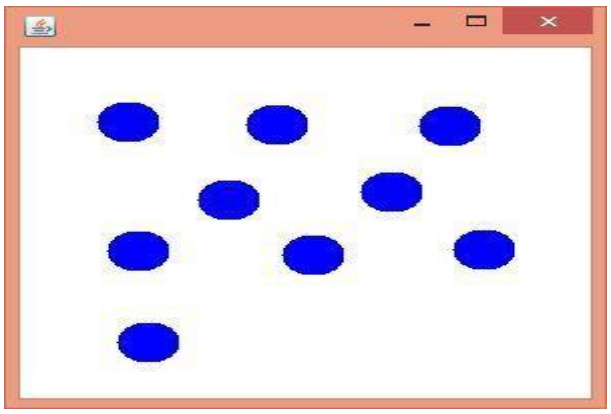
Output:

Java MouseListener Example 2

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  public class MouseListenerExample2 extends Frame implements MouseListener{
4.      MouseListenerExample2(){
5.          addMouseListener(this);
6.
7.          setSize(300,300);
8.          setLayout(null);
9.          setVisible(true);
10.     }
11.     public void mouseClicked(MouseEvent e) {
12.         Graphics g=getGraphics();
13.         g.setColor(Color.BLUE);
14.         g.fillOval(e.getX(),e.getY(),30,30);
15.     }
16.     public void mouseEntered(MouseEvent e) {}
17.     public void mouseExited(MouseEvent e) {}
18.     public void mousePressed(MouseEvent e) {}
```

```
19.    public void mouseReleased(MouseEvent e) {}
20. 20.
21. public static void main(String[] args) {
22.    new MouseListenerExample2();
23. }
24. }
```

Output:



Java 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

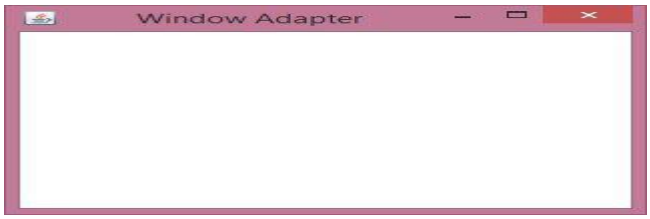
Adapter class	Listener interface
MouseInputAdapter	MouseInputListener
InternalFrameAdapter	InternalFrameListener

Java WindowAdapter Example

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

```
3. public class AdapterExample{
4.     Frame f;
5.     AdapterExample(){
6.         f=new Frame("Window Adapter");
7.         f.addWindowListener(new WindowAdapter(){
8.             public void windowClosing(WindowEvent e) {
9.                 f.dispose();
10.            }
11.        });
12.
13.        f.setSize(400,400);
14.        f.setLayout(null);
15.        f.setVisible(true);
16.    }
17. public static void main(String[] args) {
18.    new AdapterExample();
19. }
20. }
```

Output:

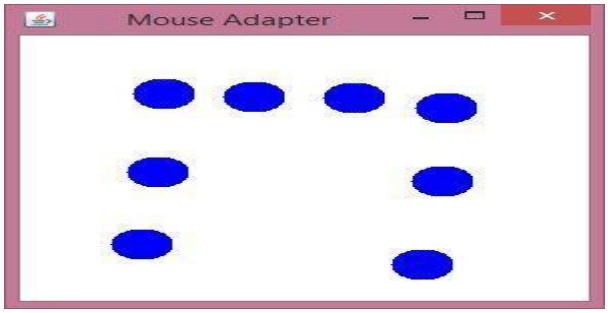


Java MouseAdapter Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseAdapterExample extends MouseAdapter{
4.     Frame f;
5.     MouseAdapterExample(){
6.         f=new Frame("Mouse Adapter");
7.         f.addMouseListener(this);
8.
9.         f.setSize(300,300);
```

```
10.         f.setLayout(null);
11.         f.setVisible(true);
12.     }
13.     public void mouseClicked(MouseEvent e) {
14.         Graphics g=f.getGraphics();
15.         g.setColor(Color.BLUE);
16.         g.fillOval(e.getX(),e.getY(),30,30);
17.     }
18.
19. public static void main(String[] args) {
20.     new MouseAdapterExample();
21. }
22. }
```

Output:

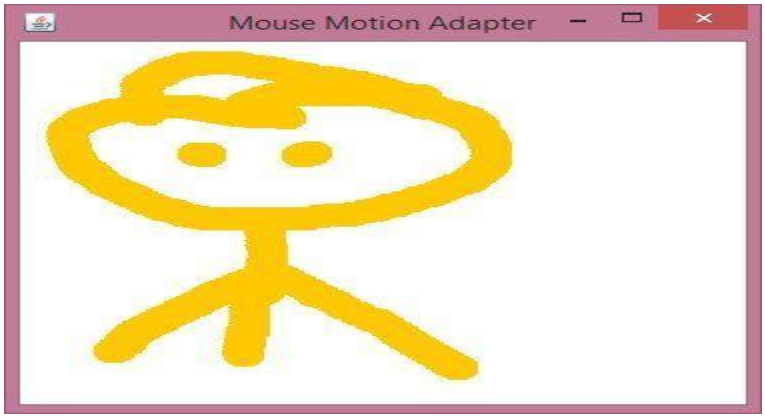


Java MouseMotionAdapter Example

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  public class MouseMotionAdapterExample extends MouseMotionAdapter{
4.      Frame f;
5.      MouseMotionAdapterExample(){
6.          f=new Frame("Mouse Motion Adapter");
7.          f.addMouseMotionListener(this);
8.
9.          f.setSize(300,300);
10.         f.setLayout(null);
11.         f.setVisible(true);
12.     }
```

```
13. public void mouseDragged(MouseEvent e) {
14.     Graphics g=f.getGraphics();
15.     g.setColor(Color.ORANGE);
16.     g.fillOval(e.getX(),e.getY(),20,20);
17. }
18. public static void main(String[] args) {
19.     new MouseMotionAdapterExample();
20. }
21. }
```

Output:

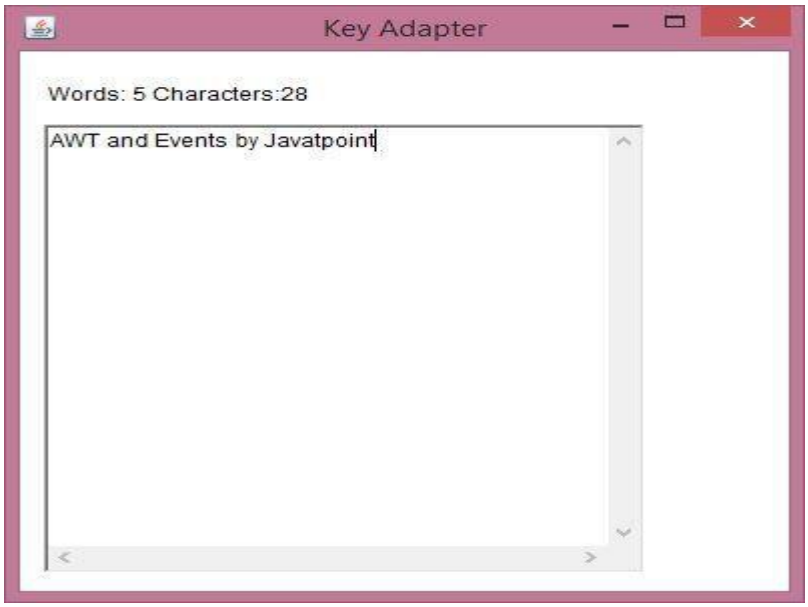


Java KeyAdapter Example

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  public class KeyAdapterExample extends KeyAdapter{
4.      Label l;
5.      TextArea area;
6.      Frame f;
7.      KeyAdapterExample(){
8.          f=new Frame("Key Adapter");
9.          l=new Label();
10.         l.setBounds(20,50,200,20);
11.         area=new TextArea();
12.         area.setBounds(20,80,300, 300);
13.         area.addKeyListener(this);
14.
```

```
15.         f.add(l);f.add(area);
16.        f.setSize(400,400);
17.         f.setLayout(null);
18.        f.setVisible(true);
19.    }
20.    public void keyReleased(KeyEvent e) {
21.         String text=area.getText();
22.        String words[]=text.split("\\s");
23.         l.setText("Words: "+words.length+" Characters:"+text.length());
24.    }
25.
26.    public static void main(String[] args) {
27.         new KeyAdapterExample();
28.    }
29. }
```

Output:



Java Inner Classes

1. Java Inner classes
2. Advantage of Inner class

3. Difference between nested class and inner class
4. Types of Nested classes

Java inner class or nested class is a class which is declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Additionally, it can access all the members of outer class including private data members and methods.

Syntax of Inner class

```
1. class Java_Outer_class{
2.  //code
3.  class Java_Inner_class{
4.   //code
5.  }
6. }
```

Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

1) Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.

2) Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.

3) **Code Optimization**: It requires less code to write.

Difference between nested class and inner class in Java

Inner class is a part of nested class. Non-static nested classes are known as inner classes.

Types of Nested classes

There are two types of nested classes non-static and static nested classes.The non-static nested classes are also known as inner classes.

- Non-static nested class (inner class)
    1. Member inner class
    2. Anonymous inner class
    3. Local inner class
- Static nested class

Type	Description
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing interface or extending class. Its name is decided by the java compiler.
Local Inner Class	A class created within method.
Static Nested Class	A static class created within class.
Nested Interface	An interface created within class or interface.

Java Anonymous inner class

A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways:

1. Class (may be abstract or concrete).
2. Interface

Java anonymous inner class example using class

```
1. abstract class Person{
```

```
2.    abstract void eat();
3. }
4.  class TestAnonymousInner{
5.   public static void main(String args[]){
6.    Person p=new Person(){
7.    void eat(){System.out.println("nice fruits");}
8.    };
9.    p.eat();
10. }
11. }
```

Test it Now

Output:

```
nice fruits
```

Internal working of given code

```
1. Person p=new Person(){
2. void eat(){System.out.println("nice fruits");}
3. };
```

1. A class is created but its name is decided by the compiler which extends the Person class and provides the implementation of the eat() method.
2. An object of Anonymous class is created that is referred by p reference variable of Person type.

Internal class generated by the compiler

```
1. import java.io.PrintStream;
2. static class TestAnonymousInner$1 extends Person
3. {
4.    TestAnonymousInner$1(){}
5.    void eat()
6.    {
7.        System.out.println("nice fruits");
8.    }
9. }
```

Java anonymous inner class example using interface

```
1. interface Eatable{
2.  void eat();
3. }
4. class TestAnnonymousInner1{
5.  public static void main(String args[]){
6.  Eatable e=new Eatable(){
7.   public void eat(){System.out.println("nice fruits");}
8.  };
9.  e.eat();
10. }
11. }
```

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

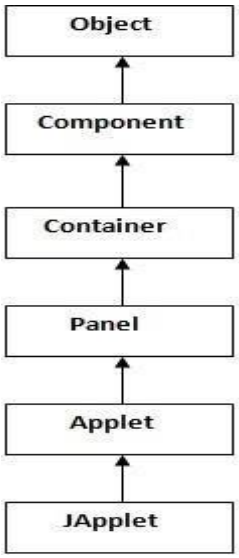
There are many advantages of applet. They are as follows:

- 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



As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

Lifecycle of Java Applet

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 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.

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **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.

1. **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.

Who is responsible to manage the life cycle of an applet?

Java Plug-in software.

How to run an Applet?

There are two ways to run an applet

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

Simple example of 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.

```
1. //First.java
2. import java.applet.Applet;
3. import java.awt.Graphics;
4. public class First extends Applet{
5.
6. public void paint(Graphics g){
7. g.drawString("welcome",150,150);
8. }
9.
10. }
```

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

myapplet.html

```
1. <html>
2. <body>
3. <applet code="First.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>
```

Simple example of 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 First.java. Now Html file is not required but it is for testing purpose only.

```
1. //First.java
2. import java.applet.Applet;
3. import java.awt.Graphics;
4. public class First extends Applet{
5.
```

```
6.  public void paint(Graphics g){
7.  g.drawString("welcome to applet",150,150);
8.  }
9.
10. }
11. /*
12. <applet code="First.class" width="300" height="300">
13. </applet>
14. */
```

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

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

Displaying Graphics in Applet

java.awt.Graphics class provides many methods for graphics programming.

Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.

8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Example of Graphics in applet:

```
1.  import java.applet.Applet;
2.  import java.awt.*;
3.
4.  public class GraphicsDemo extends Applet{
5.
6.   public void paint(Graphics g){
7.   g.setColor(Color.red);
8.   g.drawString("Welcome",50, 50);
9.  g.drawLine(20,30,20,300);
10. g.drawRect(70,100,30,30);
11. g.fillRect(170,100,30,30);
12. g.drawOval(70,200,30,30);
13.
14. g.setColor(Color.pink);
15. g.fillOval(170,200,30,30);
16. g.drawArc(90,150,30,30,30,270);
17. g.fillArc(270,150,30,30,0,180);
18.
19. }
20. }
```

myapplet.html

```
1.  <html>
```

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

Security Issues with the Applet

- **Applets** are loaded over the internet and they are prevented to make open network connection to any computer, except for the host, which provided the . ...
- They are also prevented from starting other programs on the client. ...
- **Applets** are loaded over the net. ...
- They cant load the libraries or define the native method calls.
-

Difference between a Java Application and a Java Applet

Java Application:
Java Application is just like a Java program that runs on an underlying operating system with the support of a virtual machine. It is also known as an **application program**. The graphical user interface is not necessary to execute the java applications, it can be run with or without it.
Java Applet**:**
An applet is a Java program that can be embedded into a web page. It runs inside the web browser and works at client side. An applet is embedded in an HTML page using the **APPLET** or **OBJECT** tag and hosted on a web server. Applets are used to make the web site more dynamic and entertaining.
Difference between Application and Applet:

Java Application	Java Applet
Applications are just like a Java programs that can be execute	Applets are small Java programs that are designed to be included with the HTML web

Java Application	Java Applet
independently without using the web browser.	document. They require a Java-enabled web browser for execution.
Application program requires a main function for its execution.	Applet does not require a main function for its execution.
Java application programs have the full access to the local file system and network.	Applets don't have local disk and network access.
Applications can access all kinds of resources available on the system.	Applets can only access the browser specific services. They don't have access to the local system.
Applications can executes the programs from the local system.	Applets cannot execute programs from the local machine.
An application program is needed to perform some task directly for the user.	An applet program is needed to perform small tasks or the part of it.

Parameter in Applet

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

1. **public** String getParameter(String parameterName)

Example of using parameter in Applet:

```
1. import java.applet.Applet;
2. import java.awt.Graphics;
3.
4. public class UseParam extends Applet{
5.
6.  public void paint(Graphics g){
7.  String str=getParameter("msg");
8.  g.drawString(str,50, 50);
9.  }
10.
11. }
```

myapplet.html

```
1. <html>
2. <body>
3. <applet code="UseParam.class" width="300" height="300">
4. <param name="msg" value="Welcome to applet">
5. </applet>
6. </body>
7. </html>
```

Example 2:

```
Import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet
{
        String n;
        String a;
        public void init()
        {
                n = getParameter("name");
                a = getParameter("age");
        }
        public void paint(Graphics g)
        {
                g.drawString("Name is: " + n, 20, 20);
                g.drawString("Age is: " + a, 20, 40);
        }
}
/*
        <applet code="MyApplet" height="300" width="500">
                <param name="name" value="Ramesh" />
                <param name="age" value="25" />
        </applet>
*/
```

Applet class in Applet

As we prefer Swing to AWT. Now we can use JApplet that can have all the controls of swing. The JApplet class extends the Applet class.

Example of EventHandling in JApplet:

```
1. import java.applet.*;
2. import javax.swing.*;
3. import java.awt.event.*;
4. public class EventJApplet extends JApplet implements ActionListener{
5. JButton b;
6. JTextField tf;
7. public void init(){
8.
9. tf=new JTextField();
10. tf.setBounds(30,40,150,20);
11.
12. b=new JButton("Click");
13. b.setBounds(80,150,70,40);
14.
15. add(b);add(tf);
16. b.addActionListener(this);
17.17.
18. setLayout(null);
19. }
20.
21. public void actionPerformed(ActionEvent e){
22. tf.setText("Welcome");
23. }
24. }
```

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

myapplet.html

```
1. <html>
2. <body>
3. <applet code="EventJApplet.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>
```

Painting in Applet

We can perform painting operation in applet by the mouseDragged() method of MouseMotionListener.

Example of Painting in Applet:

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.applet.*;
4. public class MouseDrag extends Applet implements MouseMotionListener{
5.
6. public void init(){
7. addMouseMotionListener(this);
8. setBackground(Color.red);
9. }
10.
11. public void mouseDragged(MouseEvent me){
12. Graphics g=getGraphics();
13. g.setColor(Color.white);
14. g.fillOval(me.getX(),me.getY(),5,5);
15. }
16. public void mouseMoved(MouseEvent me){}
17.
18. }
```

In the above example, getX() and getY() method of MouseEvent is used to get the current x-axis and y-axis. The getGraphics() method of Component class returns the object of Graphics.

myapplet.html

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

Digital clock in Applet

Digital clock can be created by using the Calendar and SimpleDateFormat class. Let's see the simple example:

Example of Digital clock in Applet:

```
import java.applet.*;
import java.awt.*;
import java.util.*;
import java.text.*;

public class DigitalClock extends Applet implements Runnable {

    Thread t = null;
    int hours=0, minutes=0, seconds=0;
    String timeString = "";
11.11.
    public void init() {
        setBackground( Color.green);
    }

    public void start() {
```

```
17.         t = new Thread( this );
18.         t.start();
19.    }
20.
21.
22.    public void run() {
23.       try {
24.          while (true) {
25.25.
26.             Calendar cal = Calendar.getInstance();
27.              hours = cal.get( Calendar.HOUR_OF_DAY );
28.             if ( hours > 12 ) hours -= 12;
29.              minutes = cal.get( Calendar.MINUTE );
30.             seconds = cal.get( Calendar.SECOND );
31.31.
32.             SimpleDateFormat formatter = new SimpleDateFormat("hh:mm:ss");
33.              Date date = cal.getTime();
34.             timeString = formatter.format( date );
35.35.
36.             repaint();
37.              t.sleep( 1000 ); // interval given in milliseconds
38.          }
39.       }
40.       catch (Exception e) { }
41.    }
42.
43.
44.    public void paint( Graphics g ) {
45.       g.setColor( Color.blue );
46.       g.drawString( timeString, 50, 50 );
47.    }
48. }
```

In the above example, getX() and getY() method of MouseEvent is used to get the current x-axis and y-axis. The getGraphics() method of Component class returns the object of Graphics.

myapplet.html

```
1. <html>
2. <body>
3. <applet code="DigitalClock.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>
```

Analog clock in Applet

Analog clock can be created by using the Math class. Let's see the simple example:

Example of Analog clock in Applet:

```
1. import java.applet.*;
2. import java.awt.*;
3. import java.util.*;
4. import java.text.*;
5.
6. public class MyClock extends Applet implements Runnable {
7.
8.     int width, height;
9.     Thread t = null;
10.    boolean threadSuspended;
11.    int hours=0, minutes=0, seconds=0;
12.    String timeString = "";
13.13.
14.    public void init() {
15.        width = getSize().width;
16.        height = getSize().height;
17.        setBackground( Color.black );
18.    }
19.
20.    public void start() {
```

```
21.      if ( t == null ) {
22.        t = new Thread( this );
23.        t.setPriority( Thread.MIN_PRIORITY );
24.        threadSuspended = false;
25.        t.start();
26.      }
27.      else {
28.        if ( threadSuspended ) {
29.           threadSuspended = false;
30.           synchronized( this ) {
31.              notify();
32.           }
33.        }
34.      }
35.   }
36.
37.   public void stop() {
38.     threadSuspended = true;
39.   }
40.
41.   public void run() {
42.     try {
43.        while (true) {
44.44.
45.           Calendar cal = Calendar.getInstance();
46.           hours = cal.get( Calendar.HOUR_OF_DAY );
47.           if ( hours > 12 ) hours -= 12;
48.           minutes = cal.get( Calendar.MINUTE );
49.           seconds = cal.get( Calendar.SECOND );
50.50.
51.           SimpleDateFormat formatter
52.             = new SimpleDateFormat( "hh:mm:ss", Locale.getDefault() );
53.           Date date = cal.getTime();
54.           timeString = formatter.format( date );
55.55.
56.           // Now the thread checks to see if it should suspend itself
```

```
57.            if ( threadSuspended ) {
58.              synchronized( this ) {
59.                  while ( threadSuspended ) {
60.                   wait();
61.                  }
62.              }
63.            }
64.           repaint();
65.            t.sleep( 1000 );  // interval specified in milliseconds
66.         }
67.      }
68.      catch (Exception e) { }
69.   }
70.
71.   void drawHand( double angle, int radius, Graphics g ) {
72.      angle -= 0.5 * Math.PI;
73.      int x = (int)( radius*Math.cos(angle) );
74.      int y = (int)( radius*Math.sin(angle) );
75.      g.drawLine( width/2, height/2, width/2 + x, height/2 + y );
76.   }
77.
78.   void drawWedge( double angle, int radius, Graphics g ) {
79.      angle -= 0.5 * Math.PI;
80.      int x = (int)( radius*Math.cos(angle) );
81.      int y = (int)( radius*Math.sin(angle) );
82.      angle += 2*Math.PI/3;
83.      int x2 = (int)( 5*Math.cos(angle) );
84.      int y2 = (int)( 5*Math.sin(angle) );
85.      angle += 2*Math.PI/3;
86.      int x3 = (int)( 5*Math.cos(angle) );
87.      int y3 = (int)( 5*Math.sin(angle) );
88.      g.drawLine( width/2+x2, height/2+y2, width/2 + x, height/2 + y );
89.      g.drawLine( width/2+x3, height/2+y3, width/2 + x, height/2 + y );
90.      g.drawLine( width/2+x2, height/2+y2, width/2 + x3, height/2 + y3 );
91.   }
92.
```

```
93.    public void paint( Graphics g ) {
94.      g.setColor( Color.gray );
95.      drawWedge( 2*Math.PI * hours / 12, width/5, g );
96.      drawWedge( 2*Math.PI * minutes / 60, width/3, g );
97.      drawHand( 2*Math.PI * seconds / 60, width/2, g );
98.      g.setColor( Color.white );
99.      g.drawString( timeString, 10, height-10 );
100.           }
101.         }
```

myapplet.html

```
1. <html>
2. <body>
3. <applet code="MyClock.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>
```