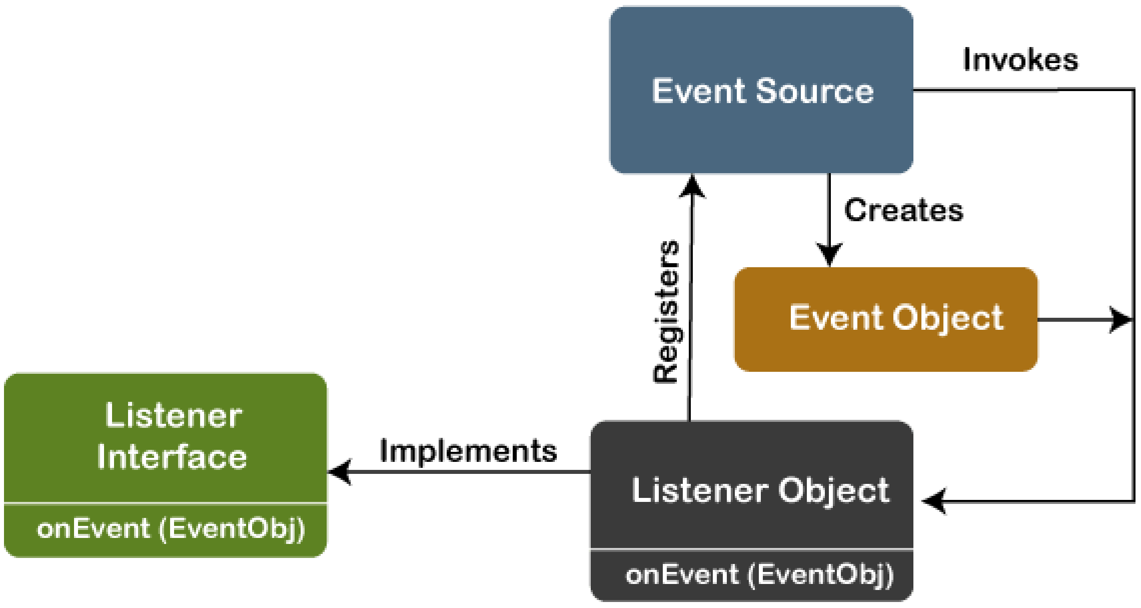
UNIT-VI

Describe Event Delegation Model

- 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.

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, below image demonstrates the event processing.



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:

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.

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.

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.

Adapter classes and inner classes with example programs

- 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**

Java WindowAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class AdapterExample{
    Frame f;
    AdapterExample(){
        f=new Frame("Window Adapter");
        f.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e) {
                f.dispose();
            }
        });

        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
public static void main(String[] args) {
    new AdapterExample();
}
}
```

Java MouseAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseAdapterExample extends MouseAdapter{
   Frame f;
   MouseAdapterExample(){
      f=new Frame("Mouse Adapter");
      f.addMouseListener(this);

      f.setSize(300,300);
      f.setLayout(null);
      f.setVisible(true);
   }
   public void mouseClicked(MouseEvent e) {
      Graphics g=f.getGraphics();
      g.setColor(Color.BLUE);
      g.fillOval(e.getX(),e.getY(),30,30);
   }

public static void main(String[] args) {
   new MouseAdapterExample();
}
}
```

Java KeyAdapter Example:

```
import java.awt.*;
import java.awt.event.*;
public class KeyAdapterExample extends KeyAdapter{
   Label l;
   TextArea area;
   Frame f;
   KeyAdapterExample(){
      f=new Frame("Key Adapter");
      l=new Label();
      l.setBounds(20,50,200,20);
      area=new TextArea();
      area.setBounds(20,80,300, 300);
      area.addKeyListener(this);

      f.add(l);f.add(area);
      f.setSize(400,400);
      f.setLayout(null);
```

```
        f.setVisible(true);
    }
    public void keyReleased(KeyEvent e) {
        String text=area.getText();
        String words[]=text.split("\\s");
        l.setText("Words: "+words.length+" Characters:"+text.length());
    }

    public static void main(String[] args) {
        new KeyAdapterExample();
    }
}
```

Java Swings

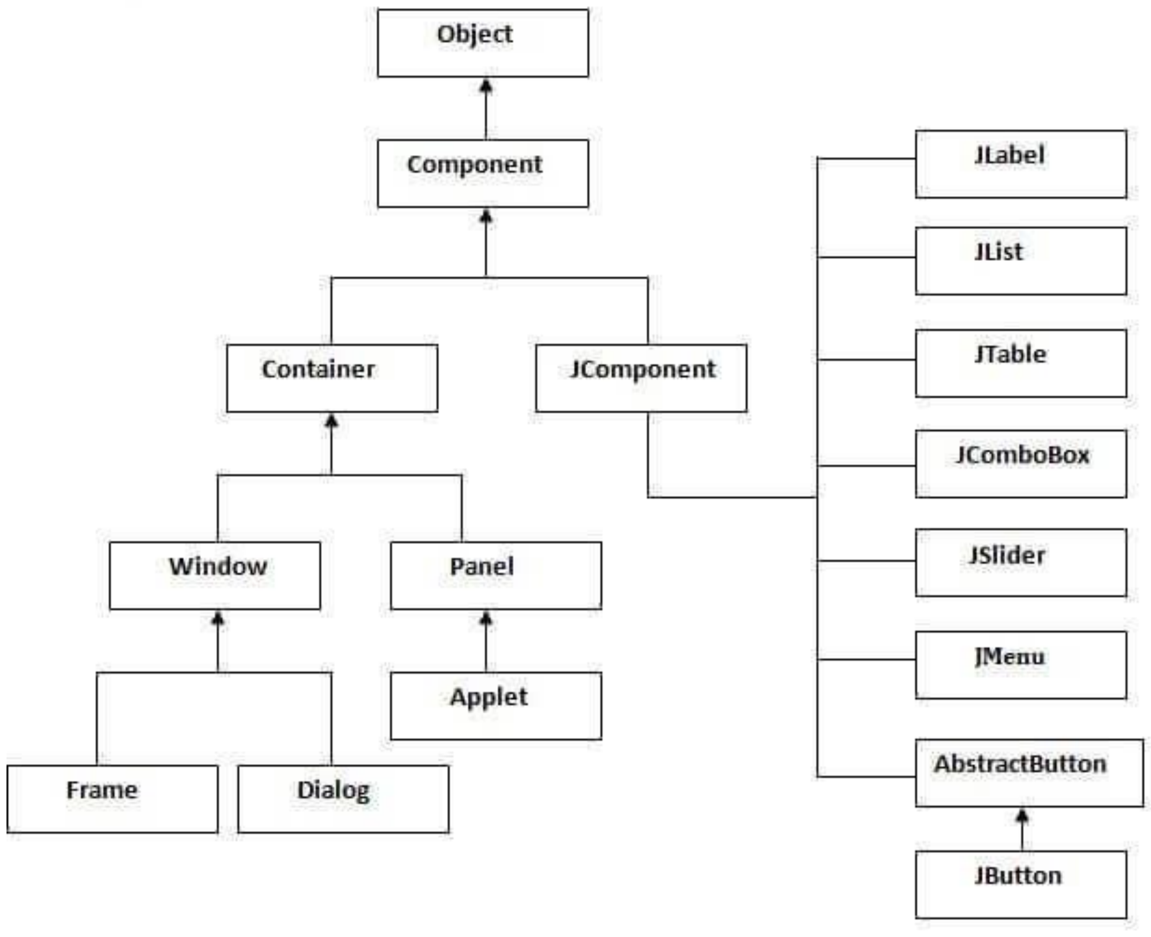
- Java Swing is a part of Java Foundation Classes (JFC) that is used to create window-based applications.
- It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
- Unlike AWT, Java Swing provides platform-independent and lightweight components.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing:

Java AWT	Java 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, scrollpanes, colorchooser, tabbedpane etc.**

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**.**

Hierarchy of Java Swing classes



Java Swing Examples

There are two ways to create a frame:

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

Example of Swing by Association inside constructor:

```
import javax.swing.*;
public class Simple {
JFrame f;
Simple(){
f=new JFrame();//creating instance of JFrame

JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);

f.add(b);//adding button in JFrame

f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}

public static void main(String[] args) {
new Simple();
}
}
```

Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

```
import javax.swing.*;
public class Simple2 extends JFrame{//inheriting JFrame
JFrame f;
Simple2(){
JButton b=new JButton("click");//create button
b.setBounds(130,100,100, 40);

add(b);//adding button on frame
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main(String[] args) {
new Simple2();
}}
```

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

public class **JButton** extends **AbstractButton**

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

```
import java.awt.event.*;
import javax.swing.*;
public class ButtonExample {
public static void main(String[] args) {
    JFrame f=new JFrame("Button Example");
    final JTextField tf=new JTextField();
    tf.setBounds(50,50, 150,20);
    JButton b=new JButton("Click Here");
    b.setBounds(50,100,95,30);
    b.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent e){
            tf.setText("Welcome to Java.");
        }
    });
    f.add(b);f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Example of displaying image on the button:

```
import javax.swing.*;
public class ButtonExample{
ButtonExample(){
JFrame f=new JFrame("Button Example");
JButton b=new JButton(new ImageIcon("D:\\icon.png"));
b.setBounds(100,100,100, 40);
f.add(b);
f.setSize(300,400);
f.setLayout(null);
f.setVisible(true);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
public static void main(String[] args) {
    new ButtonExample();
}
}
```

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

public class JLabel **extends** JComponent

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.

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	It sets the alignment of the label's contents along the

setHorizontalAlignment(int alignment)	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 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

public class JTextField **extends** JTextComponent

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 JTextArea

The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits JTextComponent class

JTextArea class declaration

public class JTextArea **extends** JTextComponent

Commonly used Constructors:

Constructor	Description
JTextArea()	Creates a text area that displays no text initially.
JTextArea(String s)	Creates a text area that displays specified text initially.

JTextArea(int row, int column)	Creates a text area with the specified number of rows and columns that displays no text initially.
JTextArea(String s, int row, int column)	Creates a text area with the specified number of rows and columns that displays specified text.

Commonly used Methods:

Methods	**Description**
void setRows(int rows)	It is used to set specified number of rows.
void setColumns(int cols)	It is used to set specified number of columns.
void setFont(Font f)	It is used to set the specified font.
void insert(String s, int position)	It is used to insert the specified text on the specified position.
void append(String s)	It is used to append the given text to the end of the document.

Java JTextArea Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class TextAreaExample implements ActionListener{
JLabel l1,l2;
JTextArea area;
JButton b;
TextAreaExample() {
    JFrame f= new JFrame();
    l1=new JLabel();
    l1.setBounds(50,25,100,30);
    l2=new JLabel();
    l2.setBounds(160,25,100,30);
    area=new JTextArea();
    area.setBounds(20,75,250,200);
    b=new JButton("Count Words");
    b.setBounds(100,300,120,30);
    b.addActionListener(this);
    f.add(l1);f.add(l2);f.add(area);f.add(b);
    f.setSize(450,450);
    f.setLayout(null);
    f.setVisible(true);
}
public void actionPerformed(ActionEvent e){
    String text=area.getText();
    String words[]=text.split("\\s");
    l1.setText("Words: "+words.length);
    l2.setText("Characters: "+text.length());
}
public static void main(String[] args) {
    new TextAreaExample();
}
}
```

Java JPasswordField

The object of a JPasswordField class is a text component specialized for password entry. It allows the editing of a single line of text. It inherits JTextField class.

JPasswordField class declaration

public class JPasswordField **extends** JTextField

Commonly used Constructors:

Constructor	**Description**
JPasswordField()	Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.
JPasswordField(int columns)	Constructs a new empty JPasswordField with the specified number of columns.

JPasswordField(String text)	Constructs a new JPasswordField initialized with the specified text.
JPasswordField(String text, int columns)	Construct a new JPasswordField initialized with the specified text and columns.

Java JPasswordField Example

Java JPasswordField Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class PasswordFieldExample {
   public static void main(String[] args) {
   JFrame f=new JFrame("Password Field Example");
    final JLabel label = new JLabel();
    label.setBounds(20,150, 200,50);
    final JPasswordField value = new JPasswordField();
    value.setBounds(100,75,100,30);
    JLabel l1=new JLabel("Username:");
      l1.setBounds(20,20, 80,30);
      JLabel l2=new JLabel("Password:");
      l2.setBounds(20,75, 80,30);
      JButton b = new JButton("Login");
      b.setBounds(100,120, 80,30);
      final JTextField text = new JTextField();
      text.setBounds(100,20, 100,30);
              f.add(value); f.add(l1); f.add(label); f.add(l2); f.add(b); f.add(text);
              f.setSize(300,300);
              f.setLayout(null);
              f.setVisible(true);
              b.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {
                 String data = "Username " + text.getText();
                 data += ", Password: "
                 + new String(value.getPassword());
                 label.setText(data);
              }
            });
}
}
```

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

public class JCheckBox **extends** JToggleButton

Commonly used Constructors:

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

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

public class JRadioButton **extends** JToggleButton

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 with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
class RadioButtonExample extends JFrame implements ActionListener{
JRadioButton rb1,rb2;
JButton b;
RadioButtonExample(){
rb1=new JRadioButton("Male");
rb1.setBounds(100,50,100,30);
rb2=new JRadioButton("Female");
rb2.setBounds(100,100,100,30);
ButtonGroup bg=new ButtonGroup();
bg.add(rb1);bg.add(rb2);
b=new JButton("click");
b.setBounds(100,150,80,30);
b.addActionListener(this);
add(rb1);add(rb2);add(b);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
if(rb1.isSelected()){
JOptionPane.showMessageDialog(this,"You are Male.");
}
if(rb2.isSelected()){
JOptionPane.showMessageDialog(this,"You are Female.");
}
}
public static void main(String args[]){
new RadioButtonExample();
}}
```

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

public class JComboBox **extends** JComponent

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 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 with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class ComboBoxExample {
JFrame f;
ComboBoxExample(){
    f=new JFrame("ComboBox Example");
    final JLabel label = new JLabel();
    label.setHorizontalAlignment(JLabel.CENTER);
    label.setSize(400,100);
    JButton b=new JButton("Show");
    b.setBounds(200,100,75,20);
    String languages[]={"C","C++","C#","Java","PHP"};
    final JComboBox cb=new JComboBox(languages);
    cb.setBounds(50, 100,90,20);
    f.add(cb); f.add(label); f.add(b);
    f.setLayout(null);
    f.setSize(350,350);
    f.setVisible(true);
    b.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
String data = "Programming language Selected: "
   + cb.getItemAt(cb.getSelectedIndex());
label.setText(data);
}
});
}
public static void main(String[] args) {
    new ComboBoxExample();
}
}
```

Java JTable

The JTable class is used to display data in tabular form. It is composed of rows and columns.

JTable class declaration

Commonly used Constructors:

Constructor	Description
JTable()	Creates a table with empty cells.
JTable(Object[][] rows, Object[] columns)	Creates a table with the specified data.

Java JTable Example

```
import javax.swing.*;
public class TableExample {
    JFrame f;
    TableExample(){
    f=new JFrame();
    String data[][]={ {"101","Amit","670000"},
                          {"102","Jai","780000"},
                          {"101","Sachin","700000"}};
    String column[]={"ID","NAME","SALARY"};
    JTable jt=new JTable(data,column);
    jt.setBounds(30,40,200,300);
    JScrollPane sp=new JScrollPane(jt);
    f.add(sp);
    f.setSize(300,400);
    f.setVisible(true);
}
public static void main(String[] args) {
    new TableExample();
}
}
```

Java JList

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

JList class declaration

```
1. public class JList extends JComponent
```

Commonly used Constructors:

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

Commonly used Methods:

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

Java JTree

The JTree class is used to display the tree structured data or hierarchical data. JTree is a complex component. It has a 'root node' at the top most which is a parent for all nodes in the tree. It inherits JComponent class.

JTree class declaration

public class JTree **extends** JComponent

Commonly used Constructors:

Constructor	Description
JTree()	Creates a JTree with a sample model.
JTree(Object [] value)	Creates a JTree with every element of the specified array as the child o new root node.
JTree(TreeNode root)	Creates a JTree with the specified TreeNode as its root, which displ the root node.

Java JTree Example

```
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;
public class TreeExample {
JFrame f;
TreeExample(){
   f=new JFrame();
   DefaultMutableTreeNode style=new DefaultMutableTreeNode("Style");
   DefaultMutableTreeNode color=new DefaultMutableTreeNode("color");
   DefaultMutableTreeNode font=new DefaultMutableTreeNode("font");
   style.add(color);
   style.add(font);
   DefaultMutableTreeNode red=new DefaultMutableTreeNode("red");
   DefaultMutableTreeNode blue=new DefaultMutableTreeNode("blue");
```

```
    DefaultMutableTreeNode black=new DefaultMutableTreeNode("black");

    DefaultMutableTreeNode green=new DefaultMutableTreeNode("green");

    color.add(red); color.add(blue); color.add(black); color.add(green);
    JTree jt=new JTree(style);
    f.add(jt);
    f.setSize(200,200);
    f.setVisible(true);
}
public static void main(String[] args) {
    new TreeExample();
}}
```

Java JOptionPane

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

JOptionPane class declaration

```
1. public class JOptionPane extends JComponent implements Accessible
```

Common Constructors of JOptionPane class

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

Common Methods of JOptionPane class

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

Java JOptionPane Example: showMessageDialog()

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    JOptionPane.showMessageDialog(f,"Hello, Welcome to Javatpoint.");
}
public static void main(String[] args) {
    new OptionPaneExample();
}
}
```

Java JOptionPane Example: showMessageDialog()

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

Java JOptionPane Example: showInputDialog()

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

Java JOptionPane Example: showConfirmDialog()

```
import javax.swing.*;
import java.awt.event.*;
public class OptionPaneExample extends WindowAdapter{
```

```
JFrame f;
OptionPaneExample(){
   f=new JFrame();
   f.addWindowListener(this);
   f.setSize(300, 300);
   f.setLayout(null);
   f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
   f.setVisible(true);
}
public void windowClosing(WindowEvent e) {
   int a=JOptionPane.showConfirmDialog(f,"Are you sure?");
if(a==JOptionPane.YES_OPTION){
   f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
public static void main(String[] args) {
   new OptionPaneExample();
}
}
```

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

```
public class JTabbedPane extends JComponent
```

Commonly used Constructors:

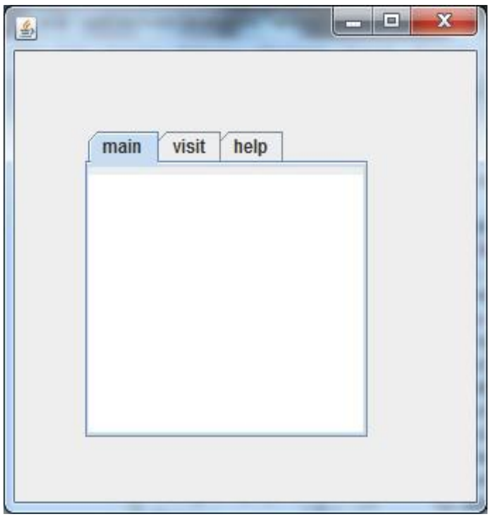
Constructor	Description
JTabbedPane()	Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top.

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

```
import javax.swing.*;
public class TabbedPaneExample {
JFrame f;
TabbedPaneExample(){
    f=new JFrame();
    JTextArea ta=new JTextArea(200,200);
    JPanel p1=new JPanel();
    p1.add(ta);
    JPanel p2=new JPanel();
    JPanel p3=new JPanel();
    JTabbedPane tp=new JTabbedPane();
    tp.setBounds(50,50,200,200);
    tp.add("main",p1);
    tp.add("visit",p2);
    tp.add("help",p3);
    f.add(tp);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
public static void main(String[] args) {
    new TabbedPaneExample();
}}
```

Output:



Java JLayeredPane

The JLayeredPane class is used to add depth to swing container. It is used to provide a third dimension for positioning component and divide the depth-range into several different layers.

JLayeredPane class declaration

```
public class JLayeredPane extends JComponent
```

Commonly used Constructors:

Constructor	Description
JLayeredPane	It is used to create a new JLayeredPane

Commonly used Methods:

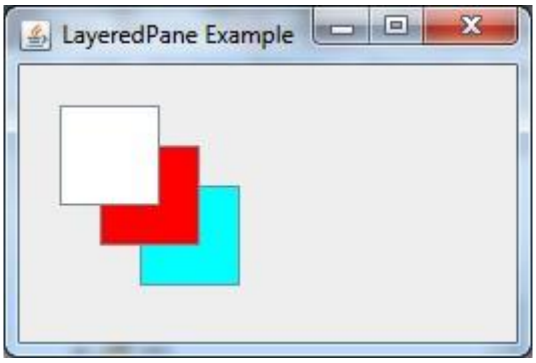
Method	Description
int getIndexOf(Component c)	It is used to return the index of the specified Component.

int getLayer(Component c)	It is used to return the layer attribute for the specified Component.
int getPosition(Component c)	It is used to return the relative position of the component within its layer.

Java JLayeredPane Example

```
import javax.swing.*;
import java.awt.*;
public class LayeredPaneExample extends JFrame {
  public LayeredPaneExample() {
  super("LayeredPane Example");
    setSize(200, 200);
    JLayeredPane pane = getLayeredPane();
    //creating buttons
    JButton top = new JButton();
    top.setBackground(Color.white);
    top.setBounds(20, 20, 50, 50);
    JButton middle = new JButton();
    middle.setBackground(Color.red);
    middle.setBounds(40, 40, 50, 50);
    JButton bottom = new JButton();
    bottom.setBackground(Color.cyan);
    bottom.setBounds(60, 60, 50, 50);
    //adding buttons on pane
    pane.add(bottom, new Integer(1));
    pane.add(middle, new Integer(2));
    pane.add(top, new Integer(3));
  }
  public static void main(String[] args) {
      LayeredPaneExample panel = new LayeredPaneExample();
      panel.setVisible(true);
  }
}
```

Output:



Java JPanel

The JPanel is a simplest container class. It provides space in which an application can attach any other component. It inherits the JComponents class.

It doesn't have title bar.

JPanel class declaration

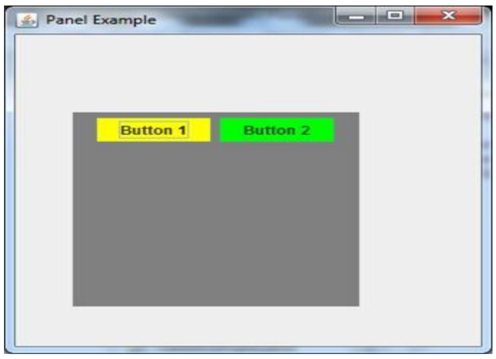
```
public class JPanel extends JComponent
```

Commonly used Constructors:

Constructor	Description
JPanel()	It is used to create a new JPanel with a double buffer and a flow layout.
JPanel(boolean isDoubleBuffered)	It is used to create a new JPanel with FlowLayout and the specified buffering strategy.
JPanel(LayoutManager layout)	It is used to create a new JPanel with the specified layout manager.

Java JPanel Example

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


Panel Example
Button 1
Button 2