UNIT IV

Introduction to JFC and Swing :

JFC stands for Java Foundation Classes, which is a set of GUI (Graphical User Interface) components for Java programmers to create sophisticated GUIs. Swing is a part of JFC that provides a rich set of GUI components, such as buttons, text fields, menus, and more, that are used to create desktop applications in Java.

Swing components are lightweight, customizable, and platform-independent, making them ideal for developing cross-platform GUI applications. Swing follows the Model-View-Controller (MVC) architecture, which separates the UI components (View) from the application logic (Model) and user interaction (Controller).

Swing provides a wide range of components and features, including:

1. Containers: Like `JFrame`, `JPanel`, `JWindow` for holding and organizing other components.

2. Components: Such as `JButton`, `JTextField`, `JLabel`, `JCheckBox`, `JRadioButton`, `JComboBox`, `JList`, `JTable`, etc., for user interaction.

3. Layout Managers: Like `FlowLayout`, `BorderLayout`, `GridLayout`, `GridBagLayout`, etc., for arranging components within containers.

4. Events and listeners: For handling user interactions and component events.

5. Graphics and painting: For custom drawing and rendering.

6. Data transfer: For drag-and-drop functionality and clipboard operations.

7. Accessibility: Features to make applications more accessible to users with disabilities.

Swing applications are written in Java and can be run on any platform that supports Java, making them highly portable.

Features of the Java Foundation Classes :

The Java Foundation Classes (JFC) is a collection of APIs that provide a rich set of features for developing desktop applications in Java. Some of the key features of JFC include:

1. Swing GUI Components: JFC includes the Swing library, which provides a comprehensive set of lightweight and customizable GUI components for building modern and interactive user interfaces.

2. Pluggable Look and Feel: Swing supports pluggable look and feel, allowing developers to change the appearance of their applications easily. Swing provides several look and feel themes, such as Metal, Windows, and Motif, among others.

3. Accessibility: JFC provides built-in support for accessibility features, making it easier for developers to create applications that are accessible to users with disabilities. Accessibility features include keyboard navigation, screen readers, and high-contrast modes.

4. Drag and Drop: JFC includes APIs for implementing drag-and-drop functionality in Java applications. Developers can enable users to drag and drop data between components or even between different applications.

5. Data Transfer: JFC provides APIs for transferring data between applications using the clipboard. Developers can use these APIs to enable copy and paste functionality in their applications.

6. Internationalization and Localization: JFC provides support for internationalization (i18n) and localization (l10n), allowing developers to create applications that can be easily translated into different languages and adapted to different locales.

7. Rich Text Support: JFC includes APIs for displaying and editing rich text, including support for fonts, styles, and colors.

8. Printing: JFC provides APIs for printing documents from Java applications, allowing developers to create print-friendly versions of their content.

Overall, the Java Foundation Classes provide a powerful and flexible framework for building desktop applications in Java, with features that help developers create applications that are both functional and user-friendly.

Swing API Components :

Swing is a part of the Java Foundation Classes (JFC) that provides a rich set of GUI components for building desktop applications in Java. Some of the key Swing components include:

1. JFrame: The main window of a Swing application. It represents a top-level window with a title bar and borders.

2. JPanel: A container that can hold other components. It is often used to organize the layout of a window.

3. JButton: A button that can be clicked by the user to trigger an action.

4. JLabel: A non-interactive component used to display text or an image.

5. JTextField: A text field that allows the user to input a single line of text.

6. JTextArea: A multi-line text area that allows the user to input or display multiple lines of text.

7. JCheckBox: A check box that allows the user to select or deselect an option.

8. JRadioButton: A radio button that allows the user to select one option from a group of options.

9. JComboBox: A drop-down combo box that allows the user to select an item from a list of options.

10. JList: A list component that displays a list of items from which the user can select one or more items.

11. JTable: A table component that displays data in rows and columns.

12. JScrollPane: A scroll pane that allows the user to scroll a component that is larger than its visible area, such as a JTextArea or JTable.

13. JMenuBar, JMenu, JMenuItem: Components for creating menus and menu items in an application.

14. JFileChooser: A component for selecting files or directories from the file system.

15. JDialog: A dialog window that is typically used for displaying messages or prompting the user for input.

These are just a few examples of the many components available in the Swing API. Swing provides a wide range of components that can be used to create complex and interactive user interfaces for desktop applications.

JComponent :

The `JComponent` class is a fundamental class in the Swing framework that serves as the base class for all Swing components. It extends the `java.awt.Container` class and provides additional functionality specific to Swing components. Here are some key aspects of the `JComponent` class:

1. Painting and Rendering: `JComponent` provides methods for customizing the appearance of a component, such as `paintComponent(Graphics g)` for custom painting and `setForeground(Color fg)` and `setBackground(Color bg)` for setting the foreground and background colors.

2. UI Delegate: `JComponent` uses a UI delegate pattern for rendering, allowing components to have different appearances on different platforms. The UI delegate is responsible for defining the look and feel of the component.

3. Event Handling: `JComponent` provides methods for handling various types of events, such as mouse events (`mouseClicked(MouseEvent e)`, `mousePressed(MouseEvent e)`, etc.), keyboard events (`keyPressed(KeyEvent e)`, `keyReleased(KeyEvent e)`, etc.), and focus events (`focusGained(FocusEvent e)`, `focusLost(FocusEvent e)`, etc.).

4. Layout Management: `JComponent` supports layout management through the `setLayout(LayoutManager layout)` method, which allows you to set the layout manager for the component.

5. Accessibility: `JComponent` includes support for accessibility features, such as keyboard navigation and screen reader compatibility, to ensure that components are usable by people with disabilities.

6. Double Buffering: `JComponent` supports double buffering, which can improve the rendering performance of components by reducing flickering when components are redrawn.

7. Component State: `JComponent` maintains various states, such as enabled/disabled, visible/hidden, and opaque/translucent, which affect how the component is rendered and interacted with.

Overall, the `JComponent` class provides a versatile and extensible framework for creating custom Swing components with rich functionality and appearance.

Containers :

In Java, containers are components used to hold and organize other components, such as buttons, text fields, and labels, in a graphical user interface (GUI). Containers are part of the AWT (Abstract Window Toolkit) and Swing libraries and provide the structure for building GUI applications. There are several types of containers available in Java, each with its own layout manager to control how components are arranged within the container. Here are some common containers in Java:

1. Panel (`java.awt.Panel`, `javax.swing.JPanel`): A generic container that can hold multiple components. Panels are often used to group related components together.

2. Frame (`java.awt.Frame`, `javax.swing.JFrame`): A top-level window with a title bar and borders. Frames are typically used as the main window for an application.

3. Dialog (`java.awt.Dialog`, `javax.swing.JDialog`): A window that is used to display messages or prompt the user for input. Dialogs are typically modal, meaning they block input to other windows in the application.

4. Window (`java.awt.Window`, `javax.swing.JWindow`): A top-level window without decorations, such as title bars or borders. Windows are often used for splash screens or custom pop-up windows.

5. Applet (`java.applet.Applet`, deprecated in favor of `javax.swing.JApplet`): A special type of container used for creating applets, which are small Java programs that run within a web browser.

6. ScrollPane (`java.awt.ScrollPane`, `javax.swing.JScrollPane`): A container that adds scroll bars to another component, such as a panel, to allow the user to scroll the contents if they do not fit within the visible area.

7. DesktopPane (`javax.swing.JDesktopPane`): A container used in a desktop application to manage internal frames (`javax.swing.JInternalFrame`), which are like independent windows within the main application window.

Each container class provides methods for adding and removing components, setting layout managers, and handling events. By using containers and layout managers effectively, you can create well-organized and visually appealing GUIs in 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.

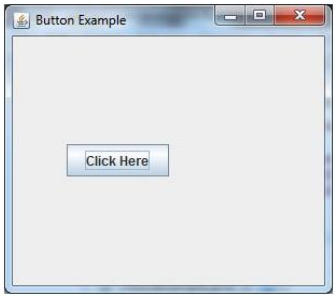
Commonly used Methods of AbstractButton class:

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

```
1. import javax.swing.*;
2. public class ButtonExample {
3. public static void main(String[] args) {
4.     JFrame f=new JFrame("Button Example");
5.     JButton b=new JButton("Click Here");
6.     b.setBounds(50,100,95,30);
7.     f.add(b);
8.     f.setSize(400,400);
9.     f.setLayout(null);
10.    f.setVisible(true);
11.}
12.}
```

Output :



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

```
1. import javax.swing.*;
2. class LabelExample
3. {
4. public static void main(String args[])
5.    {
6.    JFrame f= new JFrame("Label Example");
7.    JLabel l1,l2;
8.    l1=new JLabel("First Label.");
9.    l1.setBounds(50,50, 100,30);
10.   l2=new JLabel("Second Label.");
11.   l2.setBounds(50,100, 100,30);
12.   f.add(l1); f.add(l2);
13.   f.setSize(300,300);
14.   f.setLayout(null);
15.   f.setVisible(true);
16.   }
17.   }
```

Output :

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:

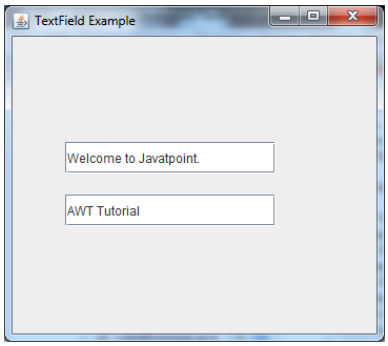
Methods	Description
void addActionListener(ActionListener l)	It is used to add the specified action listener to receive action events from this textfield.
Action getAction()	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
void setFont(Font f)	It is used to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

JTextField Example

```
1. import javax.swing.*;
2. class TextFieldExample
3. {
4. public static void main(String args[])
5.    {
6.    JFrame f= new JFrame("TextField Example");
7.    JTextField t1,t2;
8.    t1=new JTextField("Welcome to Javatpoint.");
9.    t1.setBounds(50,100, 200,30);
10.   t2=new JTextField("AWT Tutorial");
11.   t2.setBounds(50,150, 200,30);
12.   f.add(t1); f.add(t2);
13.   f.setSize(400,400);
14.   f.setLayout(null);
```

```
15.  f.setVisible(true);
16.  }
17.  }
```

Output:



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

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

```
1.  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.

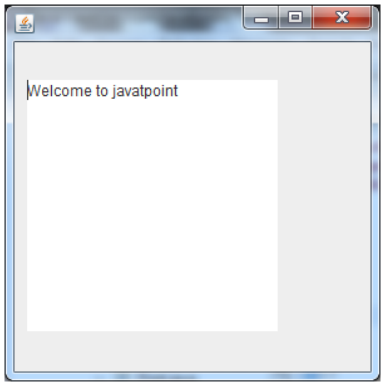
Commonly used Methods:

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.

JTextArea Example

```
1. import javax.swing.*;
2. public class TextAreaExample
3. {
4.     TextAreaExample(){
5.       JFrame f= new JFrame();
6.       JTextArea area=new JTextArea("Welcome to javatpoint");
7.       area.setBounds(10,30, 200,200);
8.       f.add(area);
9.       f.setSize(300,300);
10.      f.setLayout(null);
11.      f.setVisible(true);
12.   }
13.public static void main(String args[])
14.  {
15.  new TextAreaExample();
16.  }}
```

Output:



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

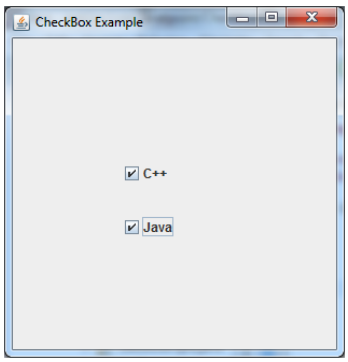
Commonly used Methods:

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

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:



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:

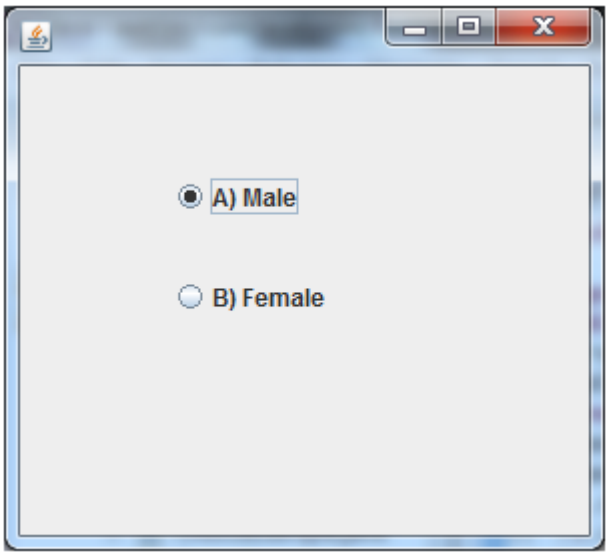
Methods	Description
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	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.

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.}
```

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

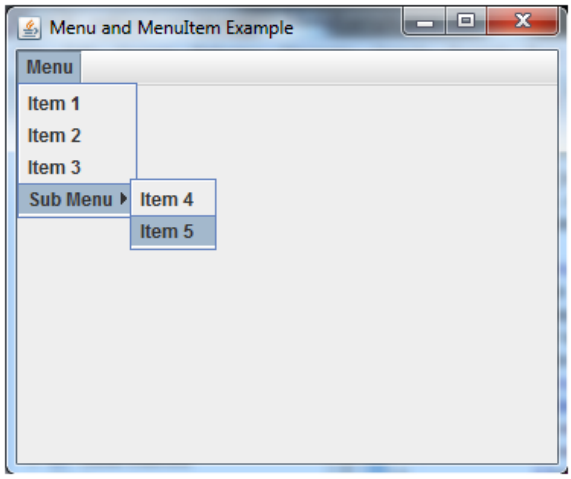
```
1. public class JMenuItem extends AbstractButton implements Accessible, MenuEle
   ment
```

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 :



Layout :

FlowLayout

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

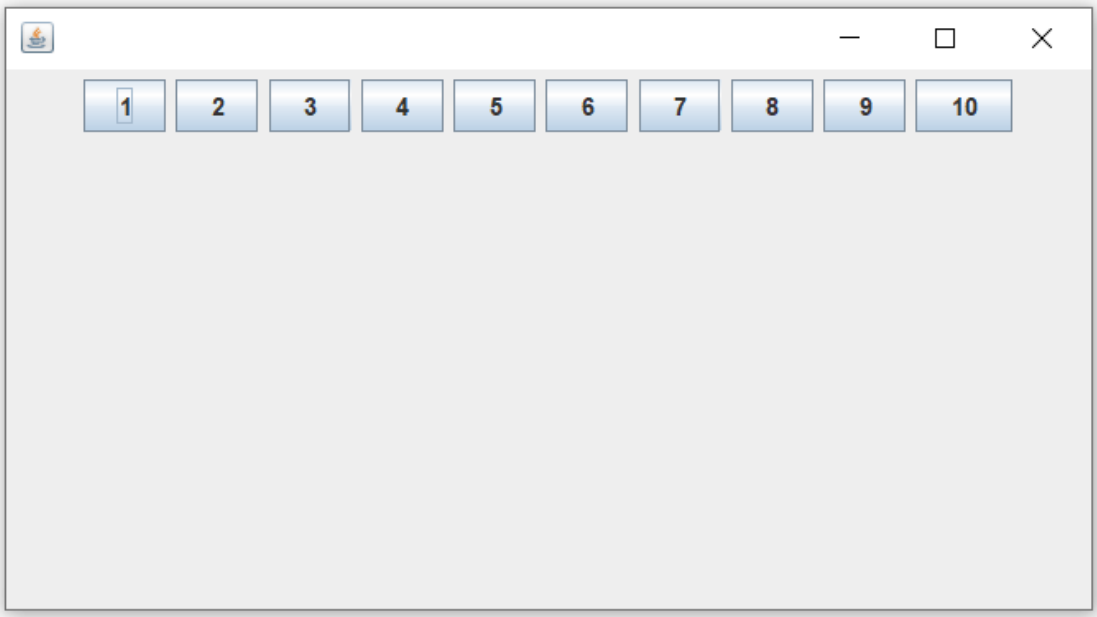
Example of FlowLayout class: Using FlowLayout() constructor

FileName: FlowLayoutExample.java

```
1. // import statements
2. import java.awt.*;
3. import javax.swing.*;
4.
5. public class FlowLayoutExample
6. {
7.
8. JFrame frameObj;
9.
10.// constructor
11.FlowLayoutExample()
12.{
13.   // creating a frame object
14.   frameObj = new JFrame();
15.
16.    // creating the buttons
17.   JButton b1 = new JButton("1");
18.   JButton b2 = new JButton("2");
19.   JButton b3 = new JButton("3");
20.   JButton b4 = new JButton("4");
21.   JButton b5 = new JButton("5");
22.   JButton b6 = new JButton("6");
```

```
23.    JButton b7 = new JButton("7");
24.    JButton b8 = new JButton("8");
25.    JButton b9 = new JButton("9");
26.    JButton b10 = new JButton("10");
27.
28.
29.    // adding the buttons to frame
30.    frameObj.add(b1); frameObj.add(b2); frameObj.add(b3); frameObj.add(b4);
31.    frameObj.add(b5); frameObj.add(b6);  frameObj.add(b7);  frameObj.add(b8);
32.    frameObj.add(b9);  frameObj.add(b10);
33.
34.    // parameter less constructor is used
35.    // therefore, alignment is center
36.    // horizontal as well as the vertical gap is 5 units.
37.    frameObj.setLayout(new FlowLayout());
38.
39.    frameObj.setSize(300, 300);
40.    frameObj.setVisible(true);
41.}
42.
43.// main method
44.public static void main(String argvs[])
45.{
46.    new FlowLayoutExample();
47.}
48.}
```

Output :



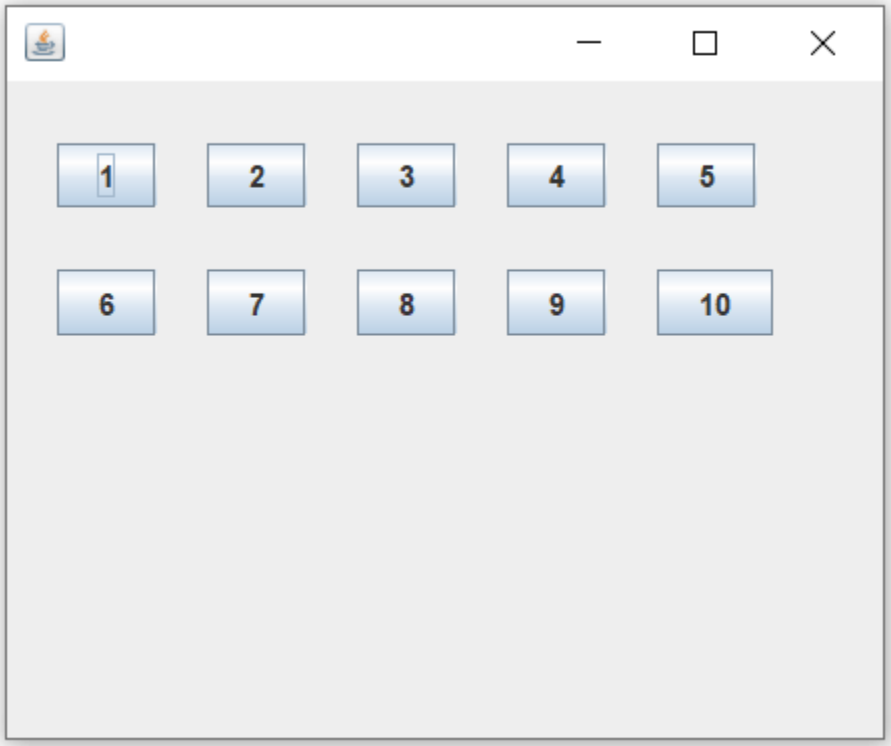
Example of FlowLayout class: Using FlowLayout(int align, int hgap, int vgap) constructor

FileName: FlowLayoutExample1.java

```
1. // import statement
2. import java.awt.*;
3. import javax.swing.*;
4.
5. public class FlowLayoutExample1
6. {
7. JFrame frameObj;
8.
9. // constructor
10.FlowLayoutExample1()
11.{
12.   // creating a frame object
13.   frameObj = new JFrame();
14.
15.    // creating the buttons
16.   JButton b1 = new JButton("1");
17.   JButton b2 = new JButton("2");
18.   JButton b3 = new JButton("3");
```

```
19.    JButton b4 = new JButton("4");
20.    JButton b5 = new JButton("5");
21.    JButton b6 = new JButton("6");
22.    JButton b7 = new JButton("7");
23.    JButton b8 = new JButton("8");
24.    JButton b9 = new JButton("9");
25.    JButton b10 = new JButton("10");
26.
27.
28.    // adding the buttons to frame
29.    frameObj.add(b1); frameObj.add(b2); frameObj.add(b3); frameObj.add(b4);
30.    frameObj.add(b5); frameObj.add(b6);  frameObj.add(b7);  frameObj.add(b8);
31.    frameObj.add(b9);  frameObj.add(b10);
32.
33.    // parameterized constructor is used
34.    // where alignment is left
35.    // horizontal gap is 20 units and vertical gap is 25 units.
36.    frameObj.setLayout(new FlowLayout(FlowLayout.LEFT, 20, 25));
37.
38.
39.    frameObj.setSize(300, 300);
40.    frameObj.setVisible(true);
41.}
42.// main method
43.public static void main(String argvs[])
44.{
45.    new FlowLayoutExample1();
46.}
47.}
```

Output:



GridLayout :

The Java GridLayout class is used to arrange the components in a 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 along with given horizontal and vertical gaps.

Example of GridLayout class: Using GridLayout() Constructor

The GridLayout() constructor creates only one row. The following example shows the usage of the parameterless constructor.

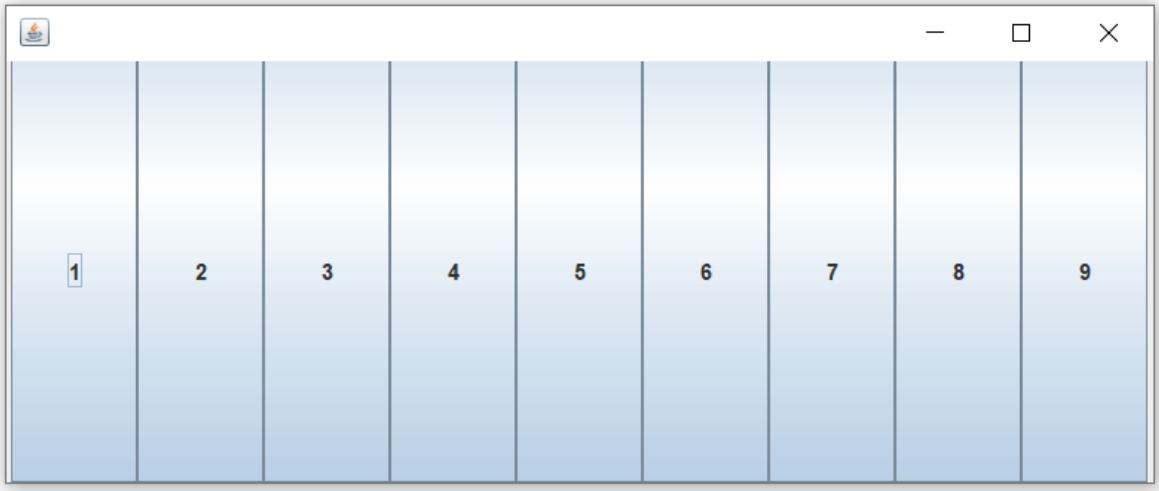
FileName: GridLayoutExample.java

```
1. // import statements
```

```
2. import java.awt.*;
3. import javax.swing.*;
4.
5. public class GridLayoutExample
6. {
7. JFrame frameObj;
8.
9. // constructor
10.GridLayoutExample()
11.{
12.frameObj = new JFrame();
13.
14.// creating 9 buttons
15.JButton btn1 = new JButton("1");
16.JButton btn2 = new JButton("2");
17.JButton btn3 = new JButton("3");
18.JButton btn4 = new JButton("4");
19.JButton btn5 = new JButton("5");
20.JButton btn6 = new JButton("6");
21.JButton btn7 = new JButton("7");
22.JButton btn8 = new JButton("8");
23.JButton btn9 = new JButton("9");
24.
25.// adding buttons to the frame
26.// since, we are using the parameterless constructor, therfore;
27.// the number of columns is equal to the number of buttons we
28.// are adding to the frame. The row count remains one.
29.frameObj.add(btn1); frameObj.add(btn2); frameObj.add(btn3);
30.frameObj.add(btn4); frameObj.add(btn5); frameObj.add(btn6);
31.frameObj.add(btn7); frameObj.add(btn8); frameObj.add(btn9);
32.
33.// setting the grid layout using the parameterless constructor
34.frameObj.setLayout(new GridLayout());
35.
36.
37.frameObj.setSize(300, 300);
38.frameObj.setVisible(true);
```

```
39.}
40.
41.// main method
42.public static void main(String argvs[])
43.{
44.new GridLayoutExample();
45.}
46.}
```

Output:



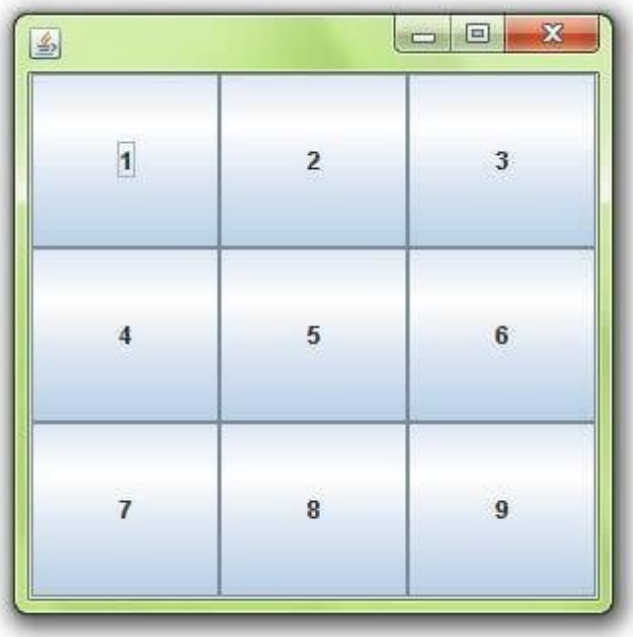
Example of GridLayout class: Using GridLayout(int rows, int columns) Constructor

FileName: MyGridLayout.java

```
1. import java.awt.*;
2. import javax.swing.*;
3. public class MyGridLayout{
4. JFrame f;
5. MyGridLayout(){
6.    f=new JFrame();
7.    JButton b1=new JButton("1");
8.    JButton b2=new JButton("2");
9.    JButton b3=new JButton("3");
10.   JButton b4=new JButton("4");
```

```
11.  JButton b5=new JButton("5");
12.  JButton b6=new JButton("6");
13.  JButton b7=new JButton("7");
14.  JButton b8=new JButton("8");
15.  JButton b9=new JButton("9");
16.   // adding buttons to the frame
17.  f.add(b1); f.add(b2); f.add(b3);
18.  f.add(b4); f.add(b5); f.add(b6);
19.  f.add(b7); f.add(b8); f.add(b9);
20.
21.  // setting grid layout of 3 rows and 3 columns
22.  f.setLayout(new GridLayout(3,3));
23.  f.setSize(300,300);
24.  f.setVisible(true);
25.}
26.public static void main(String[] args) {
27.  new MyGridLayout();
28.}
29.}
```

Output:



BorderLayout (LayoutManagers)

Java LayoutManagers

The LayoutManagers are used to arrange components in a particular manner. The **Java LayoutManagers** facilitates us to control the positioning and size of the components in GUI forms. LayoutManager is an interface that is implemented by all the classes of layout managers. There are the following classes that represent 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 a 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.
- **BorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

Example of BorderLayout class: Using BorderLayout(int hgap, int vgap) constructor

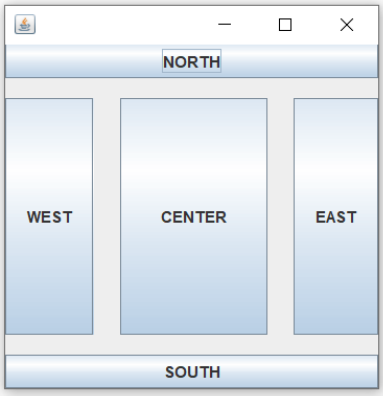
The following example inserts horizontal and vertical gaps between buttons using the parameterized constructor BorderLayout(int hgap, int gap)

FileName: BorderLayoutExample.java

```
1. // import statement
2. import java.awt.*;
3. import javax.swing.*;
4. public class BorderLayoutExample
5. {
6. JFrame jframe;
7. // constructor
8. BorderLayoutExample()
9. {
10.  // creating a Frame
11.  jframe = new JFrame();
12.  // create buttons
13.  JButton btn1 = new JButton("NORTH");
14.  JButton btn2 = new JButton("SOUTH");
15.  JButton btn3 = new JButton("EAST");
16.  JButton btn4 = new JButton("WEST");
17.  JButton btn5 = new JButton("CENTER");
18.   // creating an object of the BorderLayout class using
19.   // the parameterized constructor where the horizontal gap is 20
20.   // and vertical gap is 15. The gap will be evident when buttons are placed
21.   // in the frame
22.  jframe.setLayout(new BorderLayout(20, 15));
23.  jframe.add(btn1, BorderLayout.NORTH);
24.  jframe.add(btn2, BorderLayout.SOUTH);
25.  jframe.add(btn3, BorderLayout.EAST);
26.  jframe.add(btn4, BorderLayout.WEST);
27.  jframe.add(btn5, BorderLayout.CENTER);
28.  jframe.setSize(300,300);
29.  jframe.setVisible(true);
30.}
```

```
31. // main method
32. public static void main(String argvs[])
33. {
34.    new BorderLayoutExample();
35. }
36. }
```

Output:



Event and Listener (Java Event Handling)

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

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

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

Registration Methods

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

- **Button**
  - public void addActionListener(ActionListener a){}
- **MenuItem**
  - public void addActionListener(ActionListener a){}
- **TextField**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- **TextArea**
  - public void addTextListener(TextListener a){}

- **Checkbox**
  - public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

ActionListener Interface

The Java ActionListener is notified whenever you click on the button or menu item. It is notified against ActionEvent. The ActionListener interface is found in java.awt.event package. It has only one method: actionPerformed().

actionPerformed() method

The actionPerformed() method is invoked automatically whenever you click on the registered component.

```
1. public abstract void actionPerformed(ActionEvent e);
```

How to write ActionListener

The common approach is to implement the ActionListener. If you implement the ActionListener class, you need to follow 3 steps:

1) Implement the ActionListener interface in the class:

```
1. public class ActionListenerExample Implements ActionListener
```

2) Register the component with the Listener:

```
1. component.addActionListener(instanceOfListenerclass);
```

3) Override the actionPerformed() method:

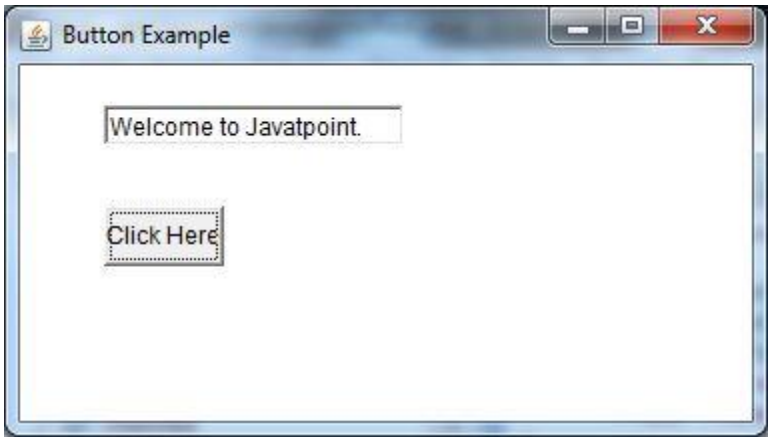
```
1. public void actionPerformed(ActionEvent e){
2.         //Write the code here
3. }
```

Java ActionListener Example: On Button click

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

```
2. import java.awt.event.*;
3. //1st step
4. public class ActionListenerExample implements ActionListener{
5. public static void main(String[] args) {
6.     Frame f=new Frame("ActionListener Example");
7.     final TextField tf=new TextField();
8.     tf.setBounds(50,50, 150,20);
9.     Button b=new Button("Click Here");
10.    b.setBounds(50,100,60,30);
11.    //2nd step
12.    b.addActionListener(this);
13.    f.add(b);f.add(tf);
14.    f.setSize(400,400);
15.    f.setLayout(null);
16.    f.setVisible(true);
17.}
18.//3rd step
19.public void actionPerformed(ActionEvent e){
20.         tf.setText("Welcome to Javatpoint.");
21.}
22.}
```

Output:



TextListener :

In Java, the `TextListener` interface is used to handle events related to text components, such as `TextField` and `TextArea`, when the text in these components changes. This interface is part of the AWT (Abstract Window Toolkit) package.

The `TextListener` interface contains a single method:

```java
public void textValueChanged(TextEvent e);
```

When a text component's text changes, it generates a `TextEvent`, and the `textValueChanged` method of any registered `TextListener` is called. This allows you to respond to changes in the text and perform actions accordingly.

Here's an example that demonstrates the use of the `TextListener` interface:

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

public class TextListenerExample extends Frame implements TextListener {
  TextField textField;

  public TextListenerExample() {
    setLayout(new FlowLayout());

    textField = new TextField(20);
    add(textField);
```

```
        // Register the TextListener
        textField.addTextListener(this);

        setSize(300, 200);
        setTitle("TextListener Example");
        setVisible(true);
    }

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

    @Override
    public void textValueChanged(TextEvent e) {
        // Handle text change event
        System.out.println("Text changed: " + textField.getText());
    }
}
```

In this example, we create a `TextField` and add a `TextListener` to it using the `addTextListener` method. The `TextListenerExample` class implements the `TextListener` interface and overrides the `textValueChanged` method to handle the text change event. When the text in the `TextField` changes, the `textValueChanged` method is called, and the new text is printed to the console.

You can use the `TextListener` interface to perform various actions based on the text input by the user, such as validation, filtering, or updating other parts of the GUI.

ItemListener Interface

The Java ItemListener is notified whenever you click on the checkbox. It is notified against ItemEvent. The ItemListener interface is found in java.awt.event package. It has only one method: itemStateChanged().

itemStateChanged() method

The itemStateChanged() method is invoked automatically whenever you click or unclick on the registered checkbox component.

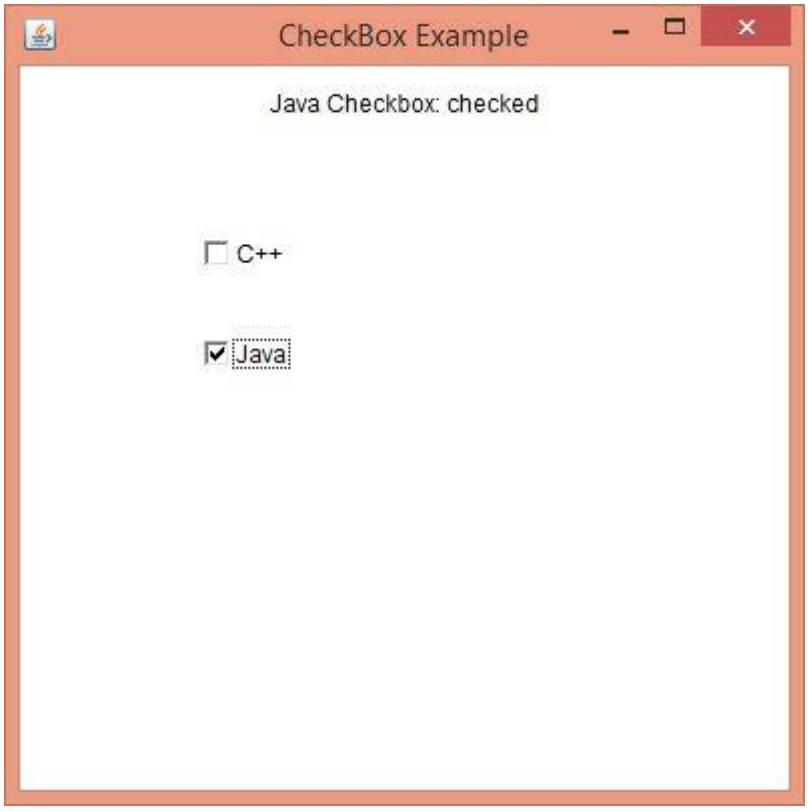
```
1. public abstract void itemStateChanged(ItemEvent e);
```

Java ItemListener Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class ItemListenerExample implements ItemListener{
4.    Checkbox checkBox1,checkBox2;
5.    Label label;
6.    ItemListenerExample(){
7.       Frame f= new Frame("CheckBox Example");
8.       label = new Label();
9.       label.setAlignment(Label.CENTER);
10.      label.setSize(400,100);
11.      checkBox1 = new Checkbox("C++");
12.      checkBox1.setBounds(100,100, 50,50);
13.      checkBox2 = new Checkbox("Java");
14.      checkBox2.setBounds(100,150, 50,50);
15.      f.add(checkBox1); f.add(checkBox2); f.add(label);
16.      checkBox1.addItemListener(this);
17.      checkBox2.addItemListener(this);
18.      f.setSize(400,400);
19.      f.setLayout(null);
20.      f.setVisible(true);
21.   }
22.   public void itemStateChanged(ItemEvent e) {
23.      if(e.getSource()==checkBox1)
```

```
24.         label.setText("C++ Checkbox: "
25.         + (e.getStateChange()==1?"checked":"unchecked"));
26.       if(e.getSource()==checkBox2)
27.       label.setText("Java Checkbox: "
28.       + (e.getStateChange()==1?"checked":"unchecked"));
29.    }
30.public static void main(String args[])
31.{
32.   new ItemListenerExample();
33.}
34.}
```

Output:



KeyListener :

The **Java KeyListener is notified whenever you change the state of key.** It is notified against KeyEvent. The KeyListener interface is found in java.awt.event package, and it has three methods.

Interface declaration

Following is the declaration for **java.awt.event.KeyListener** interface:

```
public interface KeyListener extends EventListener
```

Methods of KeyListener interface

The signature of 3 methods found in KeyListener interface are given below:

Sr. no.	Method name	Description
1.	public abstract void keyPressed (KeyEvent e);	It is invoked when a key has been pressed.
2.	public abstract void keyReleased (KeyEvent e);	It is invoked when a key has been released.
3.	public abstract void keyTyped (KeyEvent e);	It is invoked when a key has been typed.

Methods inherited

This interface inherits methods from the following interface:

Java KeyListener Example

In the following example, we are implementing the methods of the KeyListener interface.

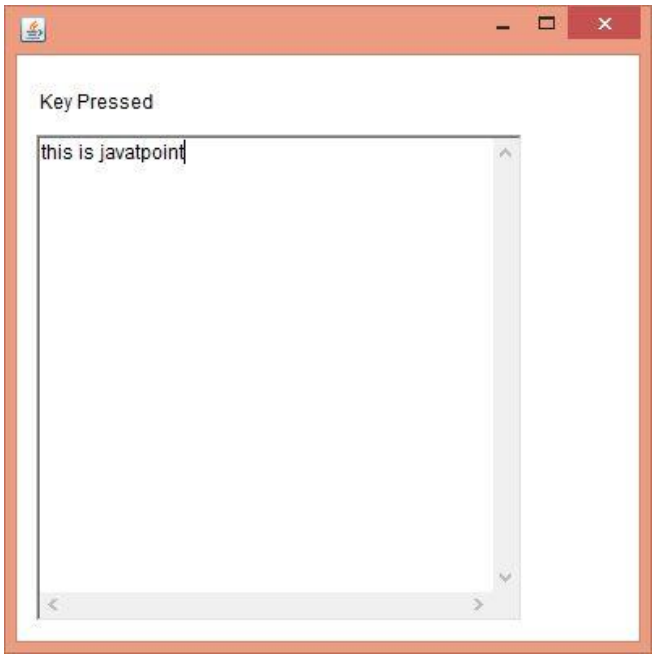
KeyListenerExample.java

```
// importing awt libraries
import java.awt.*;
import java.awt.event.*;
// class which inherits Frame class and implements KeyListener interface
public class KeyListenerExample extends Frame implements KeyListener {
// creating object of Label class   and TextArea class
 Label l;
    TextArea area;
// class constructor
```

```
10.   KeyListenerExample() {
11.        // creating the label
12.       l = new Label();
13. // setting the location of the label in frame
14.       l.setBounds (20, 50, 100, 20);
15. // creating the text area
16.       area = new TextArea();
17. // setting the location of text area
18.       area.setBounds (20, 80, 300, 300);
19. // adding the KeyListener to the text area
20.       area.addKeyListener(this);
21. // adding the label and text area to the frame
22.       add(l);
23.       add(area);
24. // setting the size, layout and visibility of frame
25.       setSize (400, 400);
26.       setLayout (null);
27.       setVisible (true);
28.   }
29. // overriding the keyPressed() method of KeyListener interface where we set the te
    xt of the label when key is pressed
30.   public void keyPressed (KeyEvent e) {
31.       l.setText ("Key Pressed");
32.   }
33. // overriding the keyReleased() method of KeyListener interface where we set the t
    ext of the label when key is released
34.   public void keyReleased (KeyEvent e) {
35.       l.setText ("Key Released");
36.   }
37. // overriding the keyTyped() method of KeyListener interface where we set the text
     of the label when a key is typed
38.   public void keyTyped (KeyEvent e) {
39.       l.setText ("Key Typed");
40.   }
41.  // main method
42.   public static void main(String[] args) {
43.       new KeyListenerExample();
```

```
44.   }
45.}
```

Output:



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:

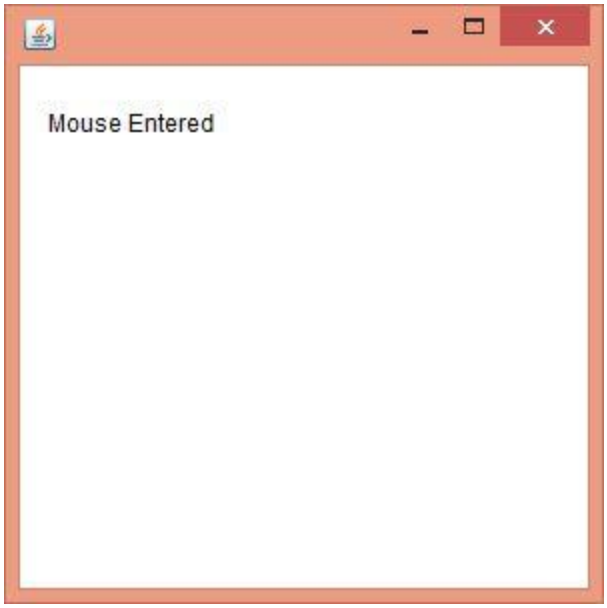
1. **public abstract void** mouseClicked(MouseEvent e);
2. **public abstract void** mouseEntered(MouseEvent e);
3. **public abstract void** mouseExited(MouseEvent e);
4. **public abstract void** mousePressed(MouseEvent e);
5. **public abstract void** mouseReleased(MouseEvent e);

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:

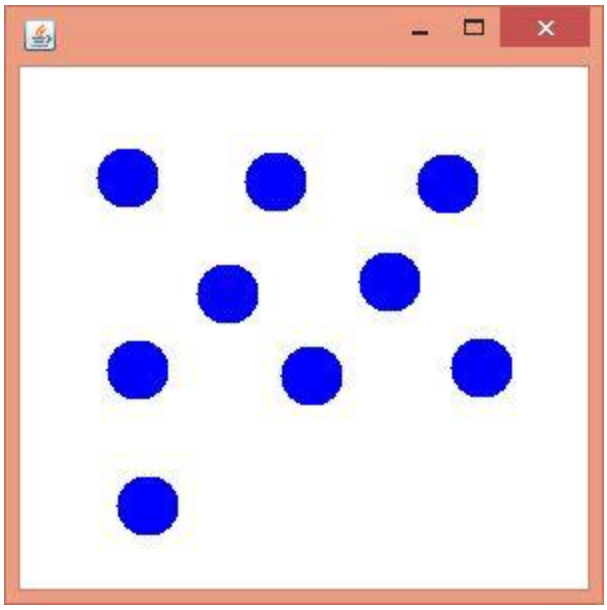


Java MouseListener Example 2

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

Output:



MouseMotionListener Interface

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

Methods of MouseMotionListener interface

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

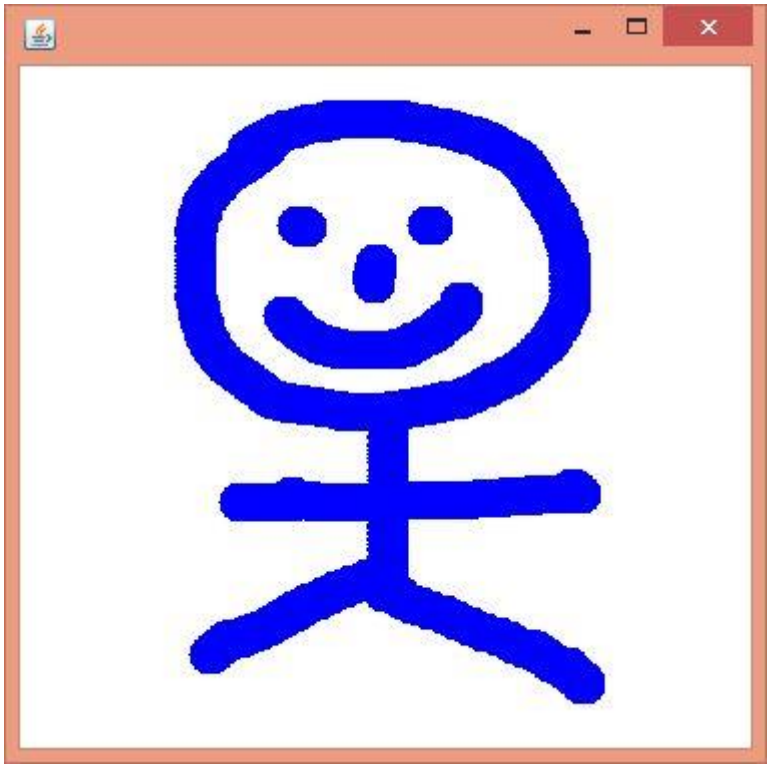
```
1. public abstract void mouseDragged(MouseEvent e);
2. public abstract void mouseMoved(MouseEvent e);
```

Java MouseMotionListener Example

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

```
3. public class MouseMotionListenerExample extends Frame implements MouseMot
   ionListener{
4.     MouseMotionListenerExample(){
5.         addMouseMotionListener(this);
6.
7.         setSize(300,300);
8.         setLayout(null);
9.         setVisible(true);
10.    }
11.public void mouseDragged(MouseEvent e) {
12.    Graphics g=getGraphics();
13.    g.setColor(Color.BLUE);
14.    g.fillOval(e.getX(),e.getY(),20,20);
15.}
16.public void mouseMoved(MouseEvent e) {}
17.
18.public static void main(String[] args) {
19.    new MouseMotionListenerExample();
20.}
21.}
```

Output:

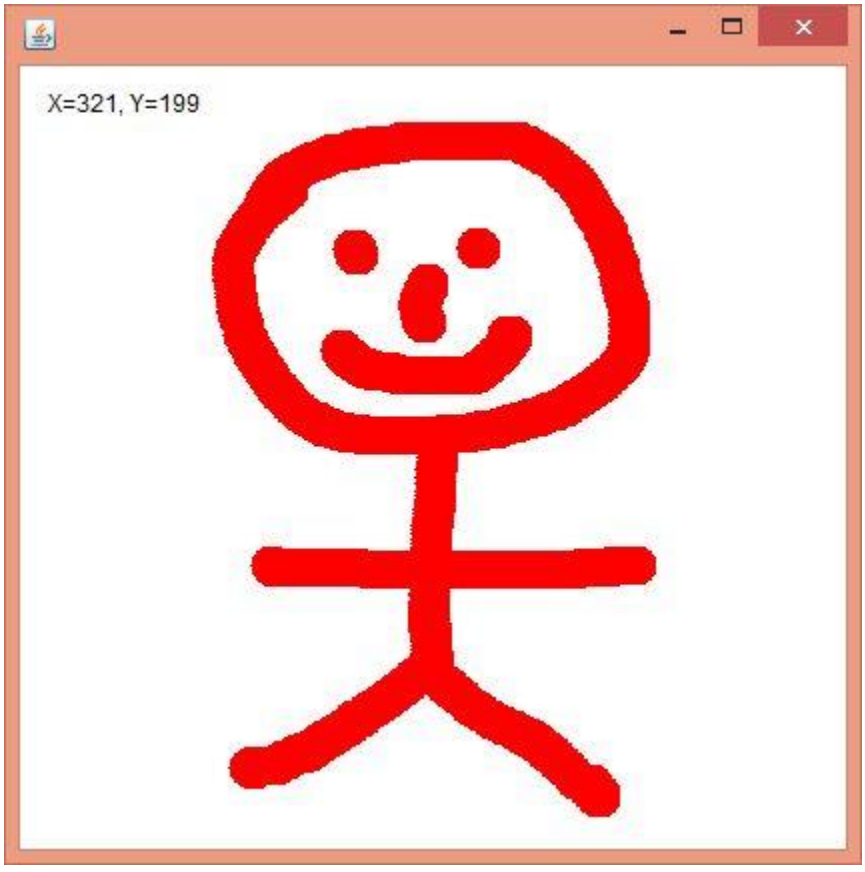


Java MouseMotionListener Example 2

```
1. import java.awt.*;
2. import java.awt.event.MouseEvent;
3. import java.awt.event.MouseMotionListener;
4. public class Paint extends Frame implements MouseMotionListener{
5.     Label l;
6.     Color c=Color.BLUE;
7.     Paint(){
8.     l=new Label();
9.     l.setBounds(20,40,100,20);
10.    add(l);
11.
12.    addMouseMotionListener(this);
13.
14.    setSize(400,400);
15.    setLayout(null);
16.    setVisible(true);
17.}
18.public void mouseDragged(MouseEvent e) {
19.    l.setText("X="+e.getX()+", Y="+e.getY());
```

```
20.    Graphics g=getGraphics();
21.    g.setColor(Color.RED);
22.    g.fillOval(e.getX(),e.getY(),20,20);
23.}
24.public void mouseMoved(MouseEvent e) {
25.    l.setText("X="+e.getX()+", Y="+e.getY());
26.}
27.public static void main(String[] args) {
28.    new Paint();
29.}
30.}
```

Output:



ItemListener Interface

The Java ItemListener is notified whenever you click on the checkbox. It is notified against ItemEvent. The ItemListener interface is found in java.awt.event package. It has only one method: itemStateChanged().

itemStateChanged() method

The itemStateChanged() method is invoked automatically whenever you click or unclick on the registered checkbox component.

```
1. public abstract void itemStateChanged(ItemEvent e);
```

Java ItemListener Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class ItemListenerExample implements ItemListener{
4.     Checkbox checkBox1,checkBox2;
5.     Label label;
6.     ItemListenerExample(){
7.         Frame f= new Frame("CheckBox Example");
8.         label = new Label();
9.         label.setAlignment(Label.CENTER);
10.        label.setSize(400,100);
11.        checkBox1 = new Checkbox("C++");
12.        checkBox1.setBounds(100,100, 50,50);
13.        checkBox2 = new Checkbox("Java");
14.        checkBox2.setBounds(100,150, 50,50);
15.        f.add(checkBox1); f.add(checkBox2); f.add(label);
16.        checkBox1.addItemListener(this);
17.        checkBox2.addItemListener(this);
18.        f.setSize(400,400);
19.        f.setLayout(null);
20.        f.setVisible(true);
21.    }
22.    public void itemStateChanged(ItemEvent e) {
23.        if(e.getSource()==checkBox1)
24.            label.setText("C++ Checkbox: "
25.            + (e.getStateChange()==1?"checked":"unchecked"));
```

```
26.         if(e.getSource()==checkBox2)
27.         label.setText("Java Checkbox: "
28.         + (e.getStateChange()==1?"checked":"unchecked"));
29.      }
30. public static void main(String args[])
31. {
32.     new ItemListenerExample();
33. }
34. }
```

Output: