JAVA SWING

java gui library

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

JAVA provides a rich set of libraries to create Graphical User Interface in a platform independent way. In this tutorial, we'll look at SWING GUI controls.

Audience

This tutorial is designed for software professionals who are willing to learn JAVA GUI Programming in simple and easy steps. This tutorial provides great understanding on JAVA GUI Programming concepts and after completing this tutorial you will be at an intermediate level of expertise, from where you can take yourself to higher levels of expertise.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of Java programming language, text editor, execution of programs, etc.

Disclaimer & Copyright

Table of Contents

1. Swing – Overview

Swing API is a set of extensible GUI Components to ease the developer's life to create JAVA based Front End/GUI Applications. It is build on top of AWT API and acts as a replacement of AWT API, since it has almost every control corresponding to AWT controls. Swing component follows a Model-View-Controller architecture to fulfill the following criterias.

- A single API is to be sufficient to support multiple look and feel.
- API is to be model driven so that the highest level API is not required to have data.
- API is to use the Java Bean model so that Builder Tools and IDE can provide better services to the developers for use.

MVC Architecture

Swing API architecture follows loosely based MVC architecture in the following manner.

- Model represents component's data.
- View represents visual representation of the component's data.
- Controller takes the input from the user on the view and reflects the changes in Component's data.
- Swing component has Model as a seperate element, while the View and Controller part are clubbed in the User Interface elements. Because of which, Swing has a pluggable look-and-feel architecture.

Swing Features

- **Light Weight** - Swing components are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls.
- **Rich Controls** - Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, colorpicker, and table controls.
- **Highly Customizable** - Swing controls can be customized in a very easy way as visual apperance is independent of internal representation.
- **Pluggable look-and-feel** - SWING based GUI Application look and feel can be changed at run-time, based on available values.

2. Swing – Environment Setup

This section guides you on how to download and set up Java on your machine. Please use the following steps to set up the environment.

Java SE is freely available from the link Download Java. Hence, you can download a version based on your operating system.

Follow the instructions to download Java and run the **.exe** to install Java on your machine. Once you have installed Java on your machine, you would need to set the environment variables to point to the correct installation directories.

Setting Up the Path for Windows 2000/XP

Assuming you have installed Java in **c:\Program Files\java\jdk** directory:

Step 1: Right-click on 'My Computer' and select 'Properties'.

Step 2: Click the 'Environment variables' button under the 'Advanced' tab.

Step 3: Alter the 'Path' variable so that it also contains the path to the Java executable. Example, if the path is currently set to **'C:\WINDOWS\SYSTEM32'**, then change your path to read **'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'**.

Setting Up the Path for Windows 95/98/ME

Assuming you have installed Java in **c:\Program Files\java\jdk** directory:

Step 1: Edit the **'C:\autoexec.bat'** file and add the following line at the end: **'SET PATH=%PATH%;C:\Program Files\java\jdk\bin'**

Setting Up the Path for Linux, UNIX, Solaris, FreeBSD

Environment variable PATH should be set to point to where the Java binaries have been installed. Refer to your Shell documentation if you have trouble doing this.

Example, if you use ***bash*** as your shell, then you would add the following line to the end **'.bashrc: export PATH=/path/to/java:$PATH'**

Popular Java Editors

To write your Java programs, you will need a text editor. There are even more sophisticated IDE available in the market. But for now, you can consider one of the following:

- **Notepad**: On Windows machine, you can use any simple text editor like Notepad (Recommended for this tutorial), TextPad.

- **Netbeans**: Netbeans is a Java IDE that is open source and free, which can be downloaded from http://www.netbeans.org/index.html.

- **Eclipse**: Eclipse is also a Java IDE developed by the Eclipse open source community and can be downloaded from http://www.eclipse.org/

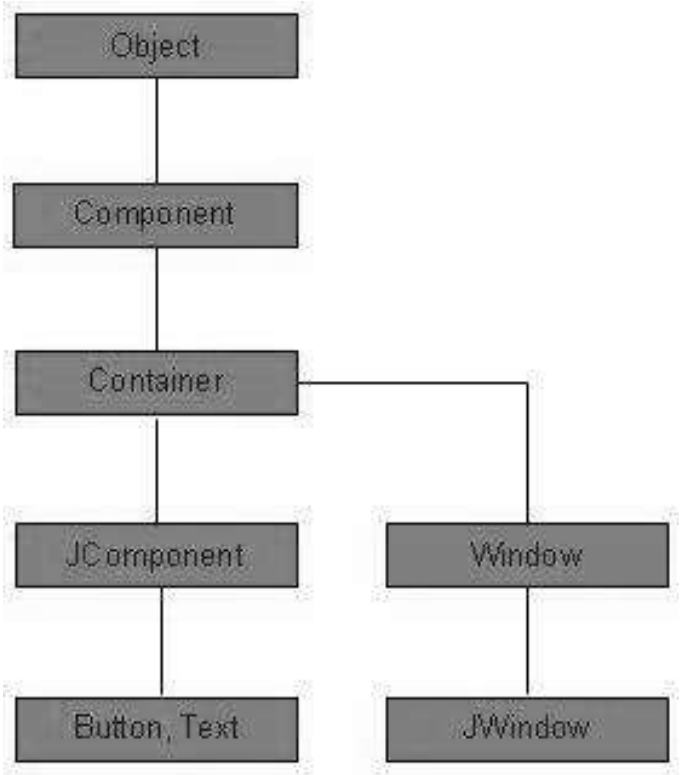
3. Swing – Controls

Every user interface considers the following three main aspects:

UI Elements: These are the core visual elements the user eventually sees and interacts with. GWT provides a huge list of widely used and common elements varying from basic to complex, which we will cover in this tutorial.

Layouts: They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface). This part will be covered in the Layout chapter.

Behavior: These are the events which occur when the user interacts with UI elements. This part will be covered in the Event Handling chapter.



Every SWING controls inherits properties from the following Component class hiearchy.

Sr. No.	Class & Description
1	**Component** A Component is the abstract base class for the non menu user-interface controls of SWING. Component represents an object with graphical representation
2	**Container** A Container is a component that can contain other SWING components

3	**JComponent** A JComponent is a base class for all SWING UI components. In order to use a SWING component that inherits from JComponent, the component must be in a containment hierarchy whose root is a top-level SWING container

Component Class

Introduction

The class **Component** is the abstract base class for the non menu user-interface controls of AWT. Component represents an object with graphical representation.

Class Declaration

Following is the declaration for **java.awt.Component** class:

```
public abstract class Component
   extends Object
      implements ImageObserver, MenuContainer, Serializable
```

Field

Following are the fields for **java.awt.Component** class:

- **static float BOTTOM_ALIGNMENT** - Ease-of-use constant for getAlignmentY.
- **static float CENTER_ALIGNMENT** - Ease-of-use constant for getAlignmentY and getAlignmentX.
- **static float LEFT_ALIGNMENT** - Ease-of-use constant for getAlignmentX.
- **static float RIGHT_ALIGNMEN**T - Ease-of-use constant for getAlignmentX.
- **static float TOP_ALIGNMENT** - Ease-of-use constant for getAlignmentY().

Class Constructors

Sr.No.	Constructor & Description
1	**protected Component()** This creates a new Component

Class Methods

Sr.No.	Method & Description
1	**boolean action(Event evt, Object what)** Deprecated. As of JDK version 1.1, should register this component as ActionListener on the component which fires action events.
2	**void add(PopupMenu popup)** Adds the specified popup menu to the component.
3	**void addComponentListener(ComponentListener l)** Adds the specified component listener to receive the component events from this component.
4	**void addFocusListener(FocusListener l)** Adds the specified focus listener to receive focus events from this component, when this component gains input focus.
5	**void addHierarchyBoundsListener(HierarchyBoundsListener l)** Adds the specified hierarchy bounds listener to receive hierarchy bounds events from this component, when the hierarchy to which this container belongs changes.
6	**void addHierarchyListener(HierarchyListener l)** Adds the specified hierarchy listener to receive hierarchy changed events from this component, when the hierarchy to which this container belongs changes.
7	**void addInputMethodListener(InputMethodListener l)** Adds the specified input method listener to receive input method events from this component.
8	**void addKeyListener(KeyListener l)** Adds the specified key listener to receive key events from this component.
9	**void addMouseListener(MouseListener l)** Adds the specified mouse listener to receive mouse events from this component.

10	**void addMouseMotionListener(MouseMotionListener l)** Adds the specified mouse motion listener to receive mouse motion events from this component.
11	**void addMouseWheelListener(MouseWheelListener l)** Adds the specified mouse wheel listener to receive mouse wheel events from this component.
12	**void addNotify()** Makes this Component displayable by connecting it to a native screen resource.
13	**void addPropertyChangeListener (PropertyChangeListener listener)** Adds a PropertyChangeListener to the listener list.
14	**void addPropertyChangeListener (String propertyName, PropertyChangeListener listener)** Adds a PropertyChangeListener to the listener list for a specific property.
15	**void applyComponentOrientation(ComponentOrientation orientation)** Sets the ComponentOrientation property of this component and all components contained within it.
16	**boolean areFocusTraversalKeysSet(int id)** Returns whether the set of focus traversal keys for the given focus traversal operation has been explicitly defined for this Component.
17	**int checkImage(Image image, ImageObserver observer)** Returns the status of the construction of a screen representation of the specified image.
18	**int checkImage(Image image,int width,int height, ImageObserver observer)** Returns the status of the construction of a screen representation of the specified image.
19	**boolean contains(int x,int y)** Checks whether this component "contains" the specified point, where **x** and **y** are defined to be relative to the coordinate system of this component.

20	**boolean contains(Point p)** Checks whether this component "contains" the specified point, where the point's **x** and **y** coordinates are defined to be relative to the coordinate system of this component.
21	**Image createImage(ImageProducer producer)** Creates an image from the specified image producer.
22	**Image createImage(int width,int height)** Creates an off-screen drawable image to be used for double buffering.
23	**VolatileImage createVolatileImage(int width,int height)** Creates a volatile off-screen drawable image to be used for double buffering.
24	**VolatileImage createVolatileImage(int width,int height, ImageCapabilities caps)** Creates a volatile off-screen drawable image, with the given capabilities.
25	**void deliverEvent(Event e)** Deprecated. As of JDK version 1.1, replaced by dispatchEvent(AWTEvent e).
26	**void disable()** Deprecated. As of JDK version 1.1, replaced by setEnabled(boolean).
27	**protected void disableEvents(long eventsToDisable)** Disables the events defined by the specified event mask parameter from being delivered to this component.
28	**void dispatchEvent(AWTEvent e)** Dispatches an event to this component or one of its sub components.
29	**void doLayout()** Prompts the layout manager to lay out this component.
30	**void enable()** Deprecated. As of JDK version 1.1, replaced by setEnabled(boolean).
31	**void enable(boolean b)**

	Deprecated. As of JDK version 1.1, replaced by setEnabled(boolean).
32	**protected void enableEvents(long eventsToEnable)** Enables the events defined by the specified event mask parameter to be delivered to this component.
33	**void enableInputMethods(boolean enable)** Enables or disables input method support for this component.
34	**protected void firePropertyChange(String propertyName, boolean oldValue, boolean newValue)** Supports reporting bound property changes for boolean properties.
35	**void firePropertyChange(String propertyName, byte oldValue, byte newValue)** Reports a bound property change.
36	**void firePropertyChange(String propertyName, char oldValue, char newValue)** Reports a bound property change.
37	**void firePropertyChange(String propertyName, double oldValue, double newValue)** Reports a bound property change.
38	**void firePropertyChange(String propertyName, float oldValue, float newValue)** Reports a bound property change.
39	**void firePropertyChange(String propertyName, long oldValue, long newValue)** Reports a bound property change.
40	**protected void firePropertyChange(String propertyName, Object oldValue, Object newValue)** Supports reporting bound property changes for Object properties.
41	**void firePropertyChange(String propertyName, short oldValue, short newValue)**

	Reports a bound property change.
42	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Component.
43	**float getAlignmentX()** Returns the alignment along the **x** axis.
44	**float getAlignmentY()** Returns the alignment along the **y** axis.
45	**Color getBackground()** Gets the background color of this component.
46	**int getBaseline(int width,int height)** Returns the baseline.
47	**Component.BaselineResizeBehavior getBaselineResizeBehavior()** Returns an enum indicating how the baseline of the component changes as the size changes.
48	**Rectangle getBounds()** Gets the bounds of this component in the form of a Rectangle object.
49	**Rectangle getBounds(Rectangle rv)** Stores the bounds of this component into "return value" rv and returns rv.
50	**ColorModel getColorModel()** Gets the instance of ColorModel used to display the component on the output device.
51	**Component getComponentAt(int x,int y)** Determines if this component or one of its immediate subcomponents contains the (**x, y**) location, and if so, returns the containing component.
52	**Component getComponentAt(Point p)** Returns the component or subcomponent that contains the specified point.

53	**ComponentListener[] getComponentListeners()** Returns an array of all the component listeners registered on this component.
54	**ComponentOrientation getComponentOrientation()** Retrieves the language-sensitive orientation that is to be used to order the elements or the text within this component.
55	**Cursor getCursor()** Gets the cursor set in the component.
56	**DropTarget getDropTarget()** Gets the DropTarget associated with this Component.
57	**Container getFocusCycleRootAncestor()** Returns the Container which is the focus cycle root of this Component's focus traversal cycle.
58	**FocusListener[] getFocusListeners()** Returns an array of all the focus listeners registered on this component.
59	**Set<AWTKeyStroke> getFocusTraversalKeys(int id)** Returns the Set of focus traversal keys for a given traversal operation for this Component.
60	**boolean getFocusTraversalKeysEnabled()** Returns whether focus traversal keys are enabled for this Component.
61	**Font getFont()** Gets the font of this component.
62	**FontMetrics getFontMetrics(Font font)** Gets the font metrics for the specified font.
63	**Color getForeground()** Gets the foreground color of this component.
64	**Graphics getGraphics()**

	Creates a graphics context for this component.
65	**GraphicsConfiguration getGraphicsConfiguration()** Gets the GraphicsConfiguration associated with this Component.
66	**int getHeight()** Returns the current height of this component.
67	**HierarchyBoundsListener[] getHierarchyBoundsListeners()** Returns an array of all the hierarchy bounds listeners registered on this component.
68	**HierarchyListener[] getHierarchyListeners()** Returns an array of all the hierarchy listeners registered on this component.
69	**boolean getIgnoreRepaint()**
70	**InputContext getInputContext()** Gets the input context used by this component for handling the communication with input methods, when the text is entered in this component.
71	**InputMethodListener[] getInputMethodListeners()** Returns an array of all the input method listeners registered on this component.
72	**InputMethodRequests getInputMethodRequests()** Gets the input method request handler which supports requests from input methods for this component.
73	**KeyListener[] getKeyListeners()** Returns an array of all the key listeners registered on this component.
74	**<T extends EventListener> T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this Component.
75	**Locale getLocale()**

	Gets the locale of this component.
76	**Point getLocation()** Gets the location of this component in the form of a point specifying the component's top-left corner.
77	**Point getLocation(Point rv)** Stores the **x,y** origin of this component into "return value" rv and returns rv.
78	**Point getLocationOnScreen()** Gets the location of this component in the form of a point specifying the component's top-left corner in the screen's coordinate space.
79	**Dimension getMaximumSize()** Gets the maximum size of this component.
80	**Dimension getMinimumSize()** Gets the mininimum size of this component.
81	**MouseListener[] getMouseListeners()** Returns an array of all the mouse listeners registered on this component.
82	**MouseMotionListener[] getMouseMotionListeners()** Returns an array of all the mouse motion listeners registered on this component.
83	**Point getMousePosition()** Returns the position of the mouse pointer in this Component's coordinate space, if the Component is directly under the mouse pointer, otherwise returns null.
84	**MouseWheelListener[] getMouseWheelListeners()** Returns an array of all the mouse wheel listeners registered on this component.
85	**String getName()** Gets the name of the component.
86	**Container getParent()**

	Gets the parent of this component.
87	**java.awt.peer.ComponentPeer getPeer()** Deprecated. As of JDK version 1.1, programs should not directly manipulate peers; replaced by boolean isDisplayable().
88	**Dimension getPreferredSize()** Gets the preferred size of this component.
89	**PropertyChangeListener[] getPropertyChangeListeners()** Returns an array of all the property change listeners registered on this component.
90	**PropertyChangeListener[] getPropertyChangeListeners(String propertyName)** Returns an array of all the listeners which have been associated with the named property.
91	**Dimension getSize()** Returns the size of this component in the form of a Dimension object.
92	**Dimension getSize(Dimension rv)** Stores the width/height of this component into "return value" rv and returns rv.
93	**Toolkit getToolkit()** Gets the toolkit of this component.
94	**Object getTreeLock()** Gets this component's locking object (the object that owns the thread sychronization monitor) for AWT component-tree and layout operations.
95	**int getWidth()** Returns the current width of this component.
96	**int getX()** Returns the current **x** coordinate of the components origin.

97	**int getY()** Returns the current **y** coordinate of the components origin.
98	**boolean gotFocus(Event evt, Object what)** Deprecated. As of JDK version 1.1, replaced by processFocusEvent(FocusEvent).
99	**boolean handleEvent(Event evt)** Deprecated. As of JDK version 1.1 replaced by processEvent(AWTEvent).
100	**boolean hasFocus()** Returns true if this Component is the focus owner.
101	**void hide()** Deprecated. As of JDK version 1.1, replaced by setVisible(boolean).
102	**boolean imageUpdate(Image img,int infoflags,int x,int y,int w,int h)** Repaints the component when the image has changed.
103	**boolean inside(int x,int y)** Deprecated. As of JDK version 1.1, replaced by contains(int, int).
104	**void invalidate()** Invalidates this component.
105	**boolean isBackgroundSet()** Returns whether the background color has been explicitly set for this Component.
106	**boolean isCursorSet()** Returns whether the cursor has been explicitly set for this Component.
107	**boolean isDisplayable()** Determines whether this component is displayable.

108	**boolean isDoubleBuffered()** Returns true if this component is painted to an offscreen image (**buffer**) that's copied to the screen later.
109	**boolean isEnabled()** Determines whether this component is enabled.
110	**boolean isFocusable()** Returns whether this Component can be focused.
111	**boolean isFocusCycleRoot(Container container)** Returns whether the specified Container is the focus cycle root of this Component's focus traversal cycle.
112	**boolean isFocusOwner()** Returns true if this Component is the focus owner.
113	**boolean isFocusTraversable()** Deprecated. As of 1.4, replaced by isFocusable().
114	**boolean isFontSet()** Returns whether the font has been explicitly set for this Component.
115	**boolean isForegroundSet()** Returns whether the foreground color has been explicitly set for this Component.
116	**boolean isLightweight()** A lightweight component doesn't have a native toolkit peer.
117	**boolean isMaximumSizeSet()** Returns true if the maximum size has been set to a non-null value otherwise returns false.

118	**boolean isMinimumSizeSet()** Returns whether or not setMinimumSize has been invoked with a non-null value.
119	**boolean isOpaque()** Returns true if this component is completely opaque, returns false by default.
120	**boolean isPreferredSizeSet()** Returns true if the preferred size has been set to a non-null value otherwise returns false.
121	**boolean isShowing()** Determines whether this component is showing on screen.
122	**boolean isValid()** Determines whether this component is valid.
123	**boolean isVisible()** Determines whether this component should be visible when its parent is visible.
124	**boolean keyDown(Event evt,int key)** Deprecated. As of JDK version 1.1, replaced by processKeyEvent(KeyEvent).
125	**boolean keyUp(Event evt,int key)** Deprecated. As of JDK version 1.1, replaced by processKeyEvent(KeyEvent).
126	**void layout()** Deprecated. As of JDK version 1.1, replaced by doLayout().
127	**void list()** Prints a listing of this component to the standard system output stream System.out.
128	**void list(PrintStream out)** Prints a listing of this component to the specified output stream.
129	**void list(PrintStream out,int indent)**

	Prints out a list, starting at the specified indentation, to the specified print stream.
130	**void list(PrintWriter out)** Prints a listing to the specified print writer.
131	**void list(PrintWriter out,int indent)** Prints out a list, starting at the specified indentation, to the specified print writer.
132	**Component locate(int x,int y)** Deprecated. As of JDK version 1.1, replaced by getComponentAt(int, int).
133	**Point location()** Deprecated. As of JDK version 1.1, replaced by getLocation().
134	**boolean lostFocus(Event evt, Object what)** Deprecated. As of JDK version 1.1, replaced by processFocusEvent(FocusEvent).
135	**boolean mouseDown(Event evt,int x,int y)** Deprecated. As of JDK version 1.1, replaced by processMouseEvent(MouseEvent).
136	**boolean mouseDrag(Event evt,int x,int y)** Deprecated. As of JDK version 1.1, replaced by processMouseMotionEvent(MouseEvent).
137	**boolean mouseEnter(Event evt,int x,int y)** Deprecated. As of JDK version 1.1, replaced by processMouseEvent(MouseEvent).

138	**boolean mouseExit(Event evt,int x,int y)** Deprecated. As of JDK version 1.1, replaced by processMouseEvent(MouseEvent).
139	**boolean mouseMove(Event evt,int x,int y)** Deprecated. As of JDK version 1.1, replaced by processMouseMotionEvent(MouseEvent).
140	**boolean mouseUp(Event evt,int x,int y)** Deprecated. As of JDK version 1.1, replaced by processMouseEvent(MouseEvent).
141	**void move(int x,int y)** Deprecated. As of JDK version 1.1, replaced by setLocation(int, int).
142	**void nextFocus()** Deprecated. As of JDK version 1.1, replaced by transferFocus().
143	**void paint(Graphics g)** Paints this component.
144	**void paintAll(Graphics g)** Paints this component and all of its subcomponents.
145	**boolean postEvent(Event e)** Deprecated. As of JDK version 1.1, replaced by dispatchEvent(AWTEvent).
146	**boolean prepareImage(Image image,int width,int height, ImageObserver observer)** Prepares an image for rendering on this component at the specified width and height.
147	**void print(Graphics g)** Prints this component.

148	**void printAll(Graphics g)** Prints this component and all of its subcomponents.
149	**protectedvoid processComponentEvent(ComponentEvent e)** Processes component events occurring on this component by dispatching them to any registered ComponentListener objects.
150	**protected void processEvent(AWTEvent e)** Processes events occurring on this component.
151	**protected void processFocusEvent(FocusEvent e)** Processes focus events occurring on this component by dispatching them to any registered FocusListener objects.
152	**protected void processHierarchyBoundsEvent(HierarchyEvent e)** Processes hierarchy bounds events occurring on this component by dispatching them to any registered HierarchyBoundsListener objects.
153	**protected void processHierarchyEvent(HierarchyEvent e)** Processes hierarchy events occurring on this component by dispatching them to any registered HierarchyListener objects.
154	**protectedvoid processInputMethodEvent(InputMethodEvent e)** Processes input method events occurring on this component by dispatching them to any registered InputMethodListener objects.
155	**protected void processKeyEvent(KeyEvent e)** Processes key events occurring on this component by dispatching them to any registered KeyListener objects.
156	**protected void processMouseEvent(MouseEvent e)** Processes mouse events occurring on this component by dispatching them to any registered MouseListener objects.

157	**protected void processMouseMotionEvent(MouseEvent e)** Processes mouse motion events occurring on this component by dispatching them to any registered MouseMotionListener objects.
158	**protected void processMouseWheelEvent(MouseWheelEvent e)** Processes mouse wheel events occurring on this component by dispatching them to any registered MouseWheelListener objects.
159	**void remove(MenuComponent popup)** Removes the specified popup menu from the component.
160	**void removeComponentListener(ComponentListener l)** Removes the specified component listener so that it no longer receives component events from this component.
161	**void removeFocusListener(FocusListener l)** Removes the specified focus listener so that it no longer receives focus events from this component.
162	**void removeHierarchyBoundsListener(HierarchyBoundsListener l)** Removes the specified hierarchy bounds listener so that it no longer receives hierarchy bounds events from this component.
163	**void removeHierarchyListener(HierarchyListener l)** Removes the specified hierarchy listener so that it no longer receives hierarchy changed events from this component.
164	**void removeInputMethodListener(InputMethodListener l)** Removes the specified input method listener so that it no longer receives input method events from this component.
165	**void removeKeyListener(KeyListener l)** Removes the specified key listener so that it no longer receives key events from this component.

166	**void removeMouseListener(MouseListener l)** Removes the specified mouse listener so that it no longer receives mouse events from this component.
167	**void removeMouseMotionListener(MouseMotionListener l)** Removes the specified mouse motion listener so that it no longer receives mouse motion events from this component.
168	**void removeMouseWheelListener(MouseWheelListener l)** Removes the specified mouse wheel listener so that it no longer receives mouse wheel events from this component.
169	**void removeNotify()** Makes this component undisplayable by destroying its native screen resource.
170	**void removePropertyChangeListener(PropertyChangeListener listener)** Removes a PropertyChangeListener from the listener list.
171	**void removePropertyChangeListener(String propertyName, PropertyChangeListener listener)** Removes a PropertyChangeListener from the listener list for a specific property.
172	**void repaint()** Repaints this component.
173	**void repaint(int x,int y,int width,int height)** Repaints the specified rectangle of this component.
174	**void repaint(long tm)** Repaints the component.
175	**void repaint(long tm,int x,int y,int width,int height)** Repaints the specified rectangle of this component within **tm** milliseconds.

176	**void requestFocus()** Requests that this component get the input focus, and that this component's top-level ancestor become the focused Window.
177	**protected boolean requestFocus(boolean temporary)** Requests that this component get the input focus, and that this component's top-level ancestor become the focused Window.
178	**boolean requestFocusInWindow()** Requests that this component get the input focus, if this component's top-level ancestor is already the focused Window.
179	**protected boolean requestFocusInWindow(boolean temporary)** Requests that this component get the input focus, if this component's top-level ancestor is already the focused Window.
180	**void reshape(int x,int y,int width,int height)** Deprecated. As of JDK version 1.1, replaced by setBounds(int, int, int, int).
181	**void resize(Dimension d)** Deprecated. As of JDK version 1.1, replaced by setSize(Dimension).
182	**void resize(int width,int height)** Deprecated. As of JDK version 1.1, replaced by setSize(int, int).
183	**void setBackground(Color c)** Sets the background color of this component.
184	**void setBounds(int x,int y,int width,int height)** Moves and resizes this component.
185	**void setBounds(Rectangle r)** Moves and resizes this component to conform to the new bounding rectangle **r**.

186	**void setComponentOrientation(ComponentOrientation o)** Sets the language-sensitive orientation that is to be used to order the elements or text within this component.
187	**void setCursor(Cursor cursor)** Sets the cursor image to the specified cursor.
188	**void setDropTarget(DropTarget dt)** Associates a DropTarget with this component.
189	**void setEnabled(boolean b)** Enables or disables this component, depending on the value of the parameter **b**.
190	**void setFocusable(boolean focusable)** Sets the focusable state of this Component to the specified value.
191	**void setFocusTraversalKeys(int id, Set<? extends AWTKeyStroke> keystrokes)** Sets the focus traversal keys for a given traversal operation for this Component.
192	**void setFocusTraversalKeysEnabled(boolean focusTraversalKeysEnabled)** Sets whether focus traversal keys are enabled for this component.
193	**void setFont(Font f)** Sets the font of this component.
194	**void setForeground(Color c)** Sets the foreground color of this component.
195	**void setIgnoreRepaint(boolean ignoreRepaint)** Sets whether or not paint messages received from the operating system should be ignored.

196	**void setLocale(Locale l)** Sets the locale of this component.
197	**void setLocation(int x,int y)** Moves this component to a new location.
198	**void setLocation(Point p)** Moves this component to a new location.
199	**void setMaximumSize(Dimension maximumSize)** Sets the maximum size of this component to a constant value.
200	**void setMinimumSize(Dimension minimumSize)** Sets the minimum size of this component to a constant value.
201	**void setName(String name)** Sets the name of the component to the specified string.
202	**void setPreferredSize(Dimension preferredSize)** Sets the preferred size of this component to a constant value.
203	**void setSize(Dimension d)** Resizes this component so that it has width **d.width** and height **d.height**.
204	**void setSize(int width,int height)** Resizes this component so that it has width **width** and height **height**.
205	**void setVisible(boolean b)** Shows or hides this component depending on the value of parameter **b**.
206	**void show()** Deprecated. As of JDK version 1.1, replaced by setVisible(boolean).
207	**void show(boolean b)** Deprecated. As of JDK version 1.1, replaced by setVisible(boolean).

208	**Dimension size()** Deprecated. As of JDK version 1.1, replaced by getSize().
209	**String toString()** Returns a string representation of this component and its values.
210	**void transferFocus()** Transfers the focus to the next component, as though this Component were the focus owner.
211	**void transferFocusBackward()** Transfers the focus to the previous component, as though this Component were the focus owner.
212	**void transferFocusUpCycle()** Transfers the focus up one focus traversal cycle.
213	**void update(Graphics g)** Updates this component.
214	**void validate()** Ensures that this component has a valid layout.
215	**Rectangle bounds()** Deprecated. As of JDK version 1.1, replaced by getBounds().
216	**protected AWTEvent coalesceEvents(AWTEvent existingEvent, AWTEvent newEvent)** Potentially coalesce an event being posted with an existing event.
217	**protected String paramString()** Returns a string representing the state of this component.
218	**protected void firePropertyChange(String propertyName,int oldValue,int newValue)** Supports reporting bound property changes for integer properties.
219	**Dimension preferredSize()**

	Deprecated. As of JDK version 1.1, replaced by getPreferredSize().
220	**boolean prepareImage(Image image, ImageObserver observer)** Prepares an image for rendering on this component.
221	**Dimension minimumSize()** Deprecated. As of JDK version 1.1, replaced by getMinimumSize().

Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

Container Class

Introduction

The class **Container** is the super class for the containers of AWT. Container object can contain other AWT components.

Class Declaration

Following is the declaration for **java.awt.Container** class:

```
public class Container
   extends Component
```

Class Constructors

Sr.No.	Constructor & Description
1	**Container()** This creates a new Container.

Class Methods

Sr.No.	Method & Description
1	**Component add(Component comp)** Appends the specified component to the end of this container.
2	**Component add(Component comp, int index)**

	Adds the specified component to this container at the given position.
3	**void add(Component comp, Object constraints)** Adds the specified component to the end of this container.
4	**void add(Component comp, Object constraints, int index)** Adds the specified component to this container with the specified constraints at the specified index.
5	**Component add(String name, Component comp)** Adds the specified component to this container.
6	**void addContainerListener(ContainerListener l)** Adds the specified container listener to receive container events from this container.
7	**protected void addImpl(Component comp, Object constraints, int index)** Adds the specified component to this container at the specified index.
8	**void addNotify()** Makes this Container displayable by connecting it to a native screen resource.
9	**void addPropertyChangeListener(PropertyChangeListener listener)** Adds a PropertyChangeListener to the listener list.
10	**void addPropertyChangeListener(String propertyName, PropertyChangeListener listener)** Adds a PropertyChangeListener to the listener list for a specific property.
11	**void applyComponentOrientation(ComponentOrientation o)** Sets the ComponentOrientation property of this container and all components contained within it.
12	**boolean areFocusTraversalKeysSet(int id)** Returns whether the Set of focus traversal keys for the given focus traversal operation has been explicitly defined for this Container.
13	**int countComponents()**

	Deprecated. As of JDK version 1.1, replaced by getComponentCount().
14	**void deliverEvent(Event e)** Deprecated. As of JDK version 1.1, replaced by dispatchEvent(AWTEvent e)
15	**void doLayout()** Causes this container to lay out its components.
16	**Component findComponentAt(int x, int y)** Locates the visible child component that contains the specified position.
17	**Component findComponentAt(Point p)** Locates the visible child component that contains the specified point.
18	**float getAlignmentX()** Returns the alignment along the **x** axis.
19	**float getAlignmentY()** Returns the alignment along the **y** axis.
20	**Component getComponent(int n)** Gets the **nt**h component in this container.
21	**Component getComponentAt(int x, int y)** Locates the component that contains the **x,y** position.
22	**Component getComponentAt(Point p)** Gets the component that contains the specified point.
23	**int getComponentCount()** Gets the number of components in this panel.
24	**Component[] getComponents()** Gets all the components in this container.
25	**int getComponentZOrder(Component comp)** Returns the **z-order** index of the component inside the container.

26	**ContainerListener[] getContainerListeners()** Returns an array of all the container listeners registered on this container.
27	**Set<AWTKeyStroke> getFocusTraversalKeys(int id)** Returns the Set of focus traversal keys for a given traversal operation for this Container.
28	**FocusTraversalPolicy getFocusTraversalPolicy()** Returns the focus traversal policy that will manage keyboard traversal of this Container's children, or null if this Container is not a focus cycle root.
29	**Insets getInsets()** Determines the insets of this container, which indicates the size of the container's border.
30	**LayoutManager getLayout()** Gets the layout manager for this container.
31	**<T extends EventListener> T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this Container.
32	**Dimension getMaximumSize()** Returns the maximum size of this container.
33	**Dimension getMinimumSize()** Returns the minimum size of this container.
34	**Point getMousePosition(boolean allowChildren)** Returns the position of the mouse pointer in this Container's coordinate space if the Container is under the mouse pointer, otherwise returns null.
35	**Dimension getPreferredSize()** Returns the preferred size of this container.
36	**Insets insets()**

	Deprecated. As of JDK version 1.1, replaced by getInsets().
37	**void invalidate()** Invalidates the container.
38	**boolean isAncestorOf(Component c)** Checks if the component is contained in the component hierarchy of this container.
39	**boolean isFocusCycleRoot()** Returns whether this Container is the root of a focus traversal cycle.
40	**boolean isFocusCycleRoot(Container container)** Returns whether the specified Container is the focus cycle root of this Container's focus traversal cycle.
41	**boolean isFocusTraversalPolicyProvider()** Returns whether this container provides focus traversal policy.
42	**boolean isFocusTraversalPolicySet()** Returns whether the focus traversal policy has been explicitly set for this Container.
43	**void layout()** Deprecated. As of JDK version 1.1, replaced by doLayout().
44	**void list(PrintStream out, int indent)** Prints a listing of this container to the specified output stream.
45	**void list(PrintWriter out, int indent)** Prints out a list, starting at the specified indentation, to the specified print writer.
46	**Component locate(int x, int y)** Deprecated. As of JDK version 1.1, replaced by getComponentAt(int, int).
47	**Dimension minimumSize()**

	Deprecated. As of JDK version 1.1, replaced by getMinimumSize().
48	**void paint(Graphics g)** Paints the container.
49	**void paintComponents(Graphics g)** Paints each of the components in this container.
50	**protected String paramString()** Returns a string representing the state of this Container.
51	**Dimension preferredSize()** Deprecated. As of JDK version 1.1, replaced by getPreferredSize().
52	**void print(Graphics g)** Prints the container.
53	**void printComponents(Graphics g)** Prints each of the components in this container.
54	**protected void processContainerEvent(ContainerEvent e)** Processes container events occurring on this container by dispatching them to any registered ContainerListener objects.
55	**protected void processEvent(AWTEvent e)** Processes events on this container.
56	**void remove(Component comp)** Removes the specified component from this container.
57	**void remove(int index)** Removes the component, specified by index, from this container.
58	**void removeAll()** Removes all the components from this container.
59	**void removeContainerListener(ContainerListener l)**

	Removes the specified container listener so it no longer receives container events from this container.
60	**void removeNotify()** Makes this container undisplayable by removing its connection to its native screen resource.
61	**void setComponentZOrder(Component comp, int index)** Moves the specified component to the specified **z-order** index in the container.
62	**void setFocusCycleRoot(boolean focusCycleRoot)** Sets whether this Container is the root of a focus traversal cycle.
63	**void setFocusTraversalKeys(int id, Set<? extends AWTKeyStroke> keystrokes)** Sets the focus traversal keys for a given traversal operation for this Container.
64	**void setFocusTraversalPolicy(FocusTraversalPolicy policy)** Sets the focus traversal policy that will manage keyboard traversal of this container's children, if this container is a focus cycle root.
65	**void setFocusTraversalPolicyProvider(boolean provider)** Sets whether this container will be used to provide focus traversal policy.
66	**void setFont(Font f)** Sets the font of this container.
67	**void setLayout(LayoutManager mgr)** Sets the layout manager for this container.
68	**void transferFocusBackward()** Transfers the focus to the previous component, as though this Component were the focus owner.
69	**void transferFocusDownCycle()** Transfers the focus down one focus traversal cycle.
70	**void update(Graphics g)**

	Updates the container.
71	**void validate()** Validates this container and all of its subcomponents.
72	**protected void validateTree()** Recursively descends the container tree and recomputes the layout for any subtrees marked as needing it (those marked as invalid).

Methods Inherited

This class inherits methods from the following classes:

- java.awt.Component
- java.lang.Object

JComponent Class

Introduction

The class **JComponent** is the base class for all Swing components except top-level containers. To use a component that inherits from JComponent, you must place the component in a containment hierarchy, whose root is a top-level SWING container.

Class Declaration

Following is the declaration for **javax.swing.JComponent** class:

```
public abstract class JComponent
   extends Container
      implements Serializable
```

Field

Following are the fields for **java.awt.Component** class:

- **protected AccessibleContext accessibleContext** - The AccessibleContext associated with this JComponent.
- **protected EventListenerList 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** - Constant used by some of the APIs to mean that no condition is defined.

- **static int WHEN_ANCESTOR_OF_FOCUSED_COMPONENT** - Constant used for registerKeyboardAction which 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** - Constant used for registerKeyboardAction which means that the command should be invoked when the component has the focus.

- **static int WHEN_IN_FOCUSED_WINDOW** - Constant used for registerKeyboardAction which means that the command should be invoked when the receiving component is in the window having the focus or is itself the focused component.

Class Constructors

Sr.No.	Constructor & Description
1	**JComponent()** Default JComponent constructor.

Class Methods

Sr.No.	Method & Description
1	**void addAncestorListener(AncestorListener listener)** Registers listener so that it will receive AncestorEvents when it or any of its ancestors move or are made visible or invisible.
2	**void addNotify()** Notifies this component that it now has a parent component.
3	**void addVetoableChangeListener(VetoableChangeListener listener)** Adds a VetoableChangeListener to the listener list.
4	**void computeVisibleRect(Rectangle visibleRect)** Returns the Component's "visible rect rectangle" - the intersection of the visible rectangles for this component and all of its ancestors.

5	**boolean contains(int x, int y)** Gives the UI delegate an opportunity to define the precise shape of this component for the sake of mouse processing.
6	**JToolTip createToolTip()** Returns the instance of JToolTip that should be used to display the tooltip.
7	**void disable()** Deprecated.As of JDK version 1.1, replaced by java.awt.Component.setEnabled(boolean).
8	**void enable()** Deprecated. As of JDK version 1.1, replaced by java.awt.Component.setEnabled(boolean).
9	**void firePropertyChange(String propertyName, boolean oldValue, boolean newValue)** Supports reporting bound property changes for boolean properties.
10	**void firePropertyChange(String propertyName, char oldValue, char newValue)** Reports a bound property change.
11	**void firePropertyChange(String propertyName, int oldValue, int newValue)** Supports reporting bound property changes for integer properties.
12	**protected void fireVetoableChange(String propertyName, Object oldValue, Object newValue)** Supports reporting constrained property changes.
13	**AccessibleContext getAccessibleContext()** Returns the AccessibleContext associated with this JComponent.
14	**ActionListener getActionForKeyStroke(KeyStroke aKeyStroke)** Returns the object that will perform the action registered for a given keystroke.

15	**ActionMap getActionMap()** Returns the ActionMap used to determine what Action to fire for particular KeyStroke binding.
16	**float getAlignmentX()** Overrides Container.getAlignmentX to return the vertical alignment.
17	**float getAlignmentY()** Overrides Container.getAlignmentY to return the horizontal alignment.
18	**AncestorListener[]getAncestorListeners()** Returns an array of all the ancestor listeners registered on this component.
19	**boolean getAutoscrolls()** Gets the autoscrolls property.
20	**int getBaseline(int width, int height)** Returns the baseline.
21	**Component.BaselineResizeBehavior getBaselineResizeBehavior()** Returns an enum indicating how the baseline of the component changes as the size changes.
22	**Border getBorder()** Returns the border of this component or null if no border is currently set.
23	**Rectangle getBounds(Rectangle rv)** Stores the bounds of this component into "return value" rv and returns rv.
24	**Object getClientProperty(Object key)** Returns the value of the property with the specified key.
25	**protected Graphics getComponentGraphics(Graphics g)** Returns the graphics object used to paint this component.

26	**JPopupMenu getComponentPopupMenu()** Returns JPopupMenu that assigned for this component.
27	**int getConditionForKeyStroke(KeyStroke aKeyStroke)** Returns the condition that determines whether a registered action occurs in response to the specified keystroke.
28	**int getDebugGraphicsOptions()** Returns the state of graphics debugging.
29	**static Locale getDefaultLocale()** Returns the default locale used to initialize each JComponent's locale property upon creation.
30	**FontMetrics getFontMetrics(Font font)** Gets the FontMetrics for the specified Font.
31	**Graphics getGraphics()** Returns this component's graphics context, which lets you draw on a component.
32	**int getHeight()** Returns the current height of this component.
33	**boolean getInheritsPopupMenu()** Returns true if the JPopupMenu should be inherited from the parent.
34	**InputMap getInputMap()** Returns the InputMap that is used when the component has focus.
35	**InputMap getInputMap(int condition)** Returns the InputMap that is used during condition.
36	**InputVerifier getInputVerifier()** Returns the input verifier for this component.

37	**Insets getInsets()** If a border has been set on this component, returns the border's insets; otherwise calls **super.getInsets**.
38	**Insets getInsets(Insets insets)** Returns an Insets object containing this component's inset values.
39	**<T extends EventListener> T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this JComponent.
40	**Point getLocation(Point rv)** Stores the **x,y** origin of this component into "return value" **rv** and returns **rv**.
41	**Dimension getMaximumSize()** If the maximum size has been set to a non-null value, just returns it.
42	**Dimension getMinimumSize()** If the minimum size has been set to a non-null value, just returns it.
43	**Component getNextFocusableComponent()** Deprecated. As of 1.4, replaced by FocusTraversalPolicy.
44	**Point getPopupLocation(MouseEvent event)** Returns the preferred location to display the popup menu in this component's coordinate system.
45	**Dimension getPreferredSize()** If the preferredSize has been set to a non-null value just returns it.
46	**KeyStroke[] getRegisteredKeyStrokes()** Returns the KeyStrokes that will initiate registered actions.
47	**JRootPane getRootPane()** Returns the JRootPane ancestor for this component.

48	**Dimension getSize(Dimension rv)** Stores the width/height of this component into "return value" **r**v and returns **rv**.
49	**Point getToolTipLocation(MouseEvent event)** Returns the tooltip location in this component's coordinate system.
50	**String getToolTipText()** Returns the tooltip string that has been set with setToolTipText.
51	**String getToolTipText(MouseEvent event)** Returns the string to be used as the tooltip for the event.
52	**Container getTopLevelAncestor()** 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.
53	**TransferHandler getTransferHandler()** Gets the transferHandler property.
54	**String getUIClassID()** Returns the UIDefaults key used to look up the name of the **swing.plaf.ComponentUI** class that defines the look and feel for this component.
55	**boolean getVerifyInputWhenFocusTarget()** Returns the value that indicates whether the input verifier for the current focus owner will be called before this component requests focus.
56	**VetoableChangeListener[] getVetoableChangeListeners()** Returns an array of all the vetoable change listeners registered on this component.

57	**Rectangle getVisibleRect()** Returns the component's "visible rectangle" - the intersection of this component's visible rectangle, new rectangle(0, 0, getWidth(), getHeight()), and all of its ancestors' visible rectangles.
58	**int getWidth()** Returns the current width of this component.
59	**int getX()** Returns the current **x** coordinate of the component's origin.
60	**int getY()** Returns the current **y** coordinate of the component's origin.
61	**void grabFocus()** Requests that this component gets the input focus, and that this component's top-level ancestor become the focused Window.
62	**boolean isDoubleBuffered()** Returns whether this component should use a buffer to paint.
63	**static boolean isLightweightComponent(Component c)** Returns true if this component is lightweight, that is, if it doesn't have a native window system peer.
64	**boolean isManagingFocus()** Deprecated.As of 1.4, replaced by Component.setFocusTraversalKeys(int, Set) and Container.setFocusCycleRoot(boolean).
65	**boolean isOpaque()** Returns true if this component is completely opaque.
66	**boolean isOptimizedDrawingEnabled()** Returns true if this component tiles its children, i.e., if it can guarantee that the children will not overlap.

67	**boolean isPaintingForPrint()** Returns true if the current painting operation on this component is part of a print operation.
68	**boolean isPaintingTile()** Returns true if the component is currently painting a tile.
69	**boolean isRequestFocusEnabled()** Returns true if this JComponent should get focus; otherwise returns false.
70	**boolean isValidateRoot()** If this method returns true, revalidate calls by descendants of this component will cause the entire tree beginning with this root to be validated.
71	**void paint(Graphics g)** Invoked by Swing to draw components.
72	**protected void paintBorder(Graphics g)** Paints the component's border.
73	**protected void paintChildren(Graphics g)** Paints this component's children.
74	**protected void paintComponent(Graphics g)** Calls the UI delegate's paint method, if the UI delegate is non-null.
75	**void paintImmediately(int x, int y, int w, int h)** Paints the specified region in this component and all of its descendants that overlap the region, immediately.
76	**void paintImmediately(Rectangle r)** Paints the specified region now.
77	**protected String paramString()** Returns a string representation of this JComponent.

78	**void print(Graphics g)** Invokes this method to print the component to the specified Graphics.
79	**void printAll(Graphics g)** Invokes this method to print the component.
80	**protected void printBorder(Graphics g)** Prints the component's border.
81	**protected void printChildren(Graphics g)** Prints this component's children.
82	**protected void printComponent(Graphics g)** This is invoked during a printing operation.
83	**protected void processComponentKeyEvent(KeyEvent e)** Processes any key events that the component itself recognizes.
84	**protected boolean processKeyBinding(KeyStroke ks, KeyEvent e, int condition, boolean pressed)** Invoked to process the key bindings for **ks** as the result of the KeyEvent **e**.
85	**protected void processKeyEvent(KeyEvent e)** Overrides processKeyEvent to process events.
86	**protected void processMouseEvent(MouseEvent e)** Processes mouse events occurring on this component by dispatching them to any registered MouseListener objects, refer to Component.processMouseEvent(MouseEvent) for a complete description of this method.
87	**protected void processMouseMotionEvent(MouseEvent e)** Processes mouse motion events, such as MouseEvent.MOUSE_DRAGGED.
88	**void putClientProperty(Object key, Object value)** Adds an arbitrary key/value "client property" to this component.

89	**void registerKeyboardAction(ActionListener anAction, KeyStroke aKeyStroke, int aCondition)** This method is now obsolete, please use a combination of getActionMap() and getInputMap() for similiar behavior.
90	**void registerKeyboardAction(ActionListener anAction, String aCommand, KeyStroke aKeyStroke, int aCondition)** This method is now obsolete, please use a combination of getActionMap() and getInputMap() for similiar behavior.
91	**void removeAncestorListener(AncestorListener listener)** Unregisters listener so that it will no longer receive AncestorEvents.
92	**void removeNotify()** Notifies this component that it no longer has a parent component.
93	**void removeVetoableChangeListener(VetoableChangeListener listener)** Removes a VetoableChangeListener from the listener list.
94	**void repaint(long tm, int x, int y, int width, int height)** Adds the specified region to the dirty region list if the component is showing.
95	**void repaint(Rectangle r)** Adds the specified region to the dirty region list, if the component is showing.
96	**boolean requestDefaultFocus()** Deprecated.As of 1.4, replaced by FocusTraversalPolicy.getDefaultComponent(Container).requestFocus()
97	**void requestFocus()** Requests that this component gets the input focus.
98	**boolean requestFocus(boolean temporary)** Requests that this component gets the input focus.

99	**boolean requestFocusInWindow()** Requests that this component gets the input focus.
100	**protected boolean requestFocusInWindow(boolean temporary)** Requests that this component gets the input focus.
101	**void resetKeyboardActions()** Unregisters all the bindings in the first tier InputMaps and ActionMap.
102	**void reshape(int x, int y, int w, int h)** Deprecated.As of JDK 5, replaced by Component.setBounds(int, int, int, int).Moves and resizes this component.
103	**void revalidate()** Supports deferred automatic layout.
104	**void scrollRectToVisible(Rectangle aRect)** Forwards the **scrollRectToVisible()** message to the JComponent's parent.
105	**void setActionMap(ActionMap am)** Sets the ActionMap to **am**.
106	**void setAlignmentX(float alignmentX)** Sets the the vertical alignment.
107	**void setAlignmentY(float alignmentY)** Sets the the horizontal alignment.
108	**void setAutoscrolls(boolean autoscrolls)** Sets the autoscrolls property.
109	**void setBackground(Color bg)** Sets the background color of this component.

110	**void setBorder(Border border)** Sets the border of this component.
111	**void setComponentPopupMenu(JPopupMenu popup)** Sets the JPopupMenu for this JComponent.
112	**void setDebugGraphicsOptions(int debugOptions)** Enables or disables diagnostic information about every graphics operation performed within the component or one of its children.
113	**static void setDefaultLocale(Locale l)** Sets the default locale used to initialize each JComponent's locale property upon creation.
114	**void setDoubleBuffered(boolean aFlag)** Sets whether this component should use a buffer to paint.
115	**void setEnabled(boolean enabled)** Sets whether or not this component is enabled.
116	**void setFocusTraversalKeys(int id, Set<? extends AWTKeyStroke> keystrokes)** Sets the focus traversal keys for a given traversal operation for this component.
117	**void setFont(Font font)** Sets the font for this component.
118	**void setForeground(Color fg)** Sets the foreground color of this component.
119	**void setInheritsPopupMenu(boolean value)** Sets whether or not getComponentPopupMenu should delegate to the parent, if this component does not have a JPopupMenu assigned to it.

120	**void setInputMap(int condition, InputMap map)** Sets the InputMap to use under the **condition** condition to map.
121	**void setInputVerifier(InputVerifier inputVerifier)** Sets the input verifier for this component.
122	**void setMaximumSize(Dimension maximumSize)** Sets the maximum size of this component to a constant value.
123	**void setMinimumSize(Dimension minimumSize)** Sets the minimum size of this component to a constant value.
124	**void setNextFocusableComponent(Component aComponent)** Deprecated. As of 1.4, replaced by FocusTraversalPolicy
125	**void setOpaque(boolean isOpaque)** If true, the component paints every pixel within its bounds.
126	**void setPreferredSize(Dimension preferredSize)** Sets the preferred size of this component.
127	**void setRequestFocusEnabled(boolean requestFocusEnabled)** Provides a hint as to whether or not this JComponent should get focus.
128	**void setToolTipText(String text)** Registers the text to display in a tool tip.
129	**void setTransferHandler(TransferHandler newHandler)** Sets the transferHandler property, which is null if the component does not support data transfer operations.
130	**protected void setUI(ComponentUI newUI)** Sets the look and feel delegate for this component.

131	**void setVerifyInputWhenFocusTarget(boolean verifyInputWhenFocusTarget)** Sets the value to indicate whether the input verifier for the current focus owner will be called before this component requests focus.
132	**void setVisible(boolean aFlag)** Makes the component visible or invisible.
133	**void unregisterKeyboardAction(KeyStroke aKeyStroke)** This method is now obsolete.
134	**void update(Graphics g)** Calls paint.
135	**void updateUI()** Resets the UI property to a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes:

- java.awt.Container
- java.awt.Component
- java.lang.Object

SWING UI Elements

Following is the list of commonly used controls while designing GUI using SWING.

Sr. No.	Control & Description
1	**JLabel** A JLabel object is a component for placing text in a container.
2	**JButton** This class creates a labeled button.
3	**JColorChooser**

	A JColorChooser provides a pane of controls designed to allow a user to manipulate and select a color.
4	**JCheck Box** A JCheckBox is a graphical component that can be in either an **on**(true) or **off** (false) state.
5	**JRadioButton** The JRadioButton class is a graphical component that can be in either an **on** (true) or **off** (false) state. in a group.
6	**JList** A JList component presents the user with a scrolling list of text items.
7	**JComboBox** A JComboBox component presents the user with a to show up menu of choices.
8	**JTextField** A JTextField object is a text component that allows editing of a single line of text.
9	**JPasswordField** A JPasswordField object is a text component specialized for password entry.
10	**JTextArea** A JTextArea object is a text component that allows editing of a multiple lines of text.
11	**ImageIcon** A ImageIcon control is an implementation of the Icon interface that paints Icons from Images.
12	**JScrollbar** A Scrollbar control represents a scroll bar component in order to enable the user to select from a range of values.
13	**JOptionPane**

	JOptionPane provides a set of standard dialog boxes that prompt the users for a value or informs them of something.
14	**JFileChooser** A JFileChooser control represents a dialog window from which the user can select a file.
15	**JProgressBar** As the task progresses towards completion, the progress bar displays the task's percentage of completion.
16	**JSlider** A JSlider lets the user graphically select a value by sliding a knob within a bounded interval.
17	**JSpinner** A JSpinner is a single line input field that lets the user select a number or an object value from an ordered sequence.

Jlabel Class

Introduction

The class **JLabel** can display either text, an image, or both. Label's contents are aligned by setting the vertical and horizontal alignment in its display area. By default, labels are vertically centered in their display area. Text-only labels are leading edge aligned, by default; image-only labels are horizontally centered, by default.

Class Declaration

Following is the declaration for **javax.swing.JLabel** class:

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

Field

Following are the fields for **javax.swing.JLabel** class:

- **protected Component labelFor**

Class Constructors

Sr.No.	Constructor & Description
1	**JLabel()** Creates a JLabel instance with no image and with an empty string for the title.
2	**JLabel(Icon image)** Creates a JLabel instance with the specified image.
3	**JLabel(Icon image, int horizontalAlignment)** Creates a JLabel instance with the specified image and horizontal alignment.
4	**JLabel(String text)** Creates a JLabel instance with the specified text.
5	**JLabel(String text, Icon icon, int horizontalAlignment)** Creates a JLabel instance with the specified text, image, and horizontal alignment.
6	**JLabel(String text, int horizontalAlignment)** Creates a JLabel instance with the specified text and horizontal alignment.

Class Methods

Sr.No.	Method & Description
1	**protected int checkHorizontalKey(int key, String message)** Verify that key is a legal value for the horizontalAlignment properties.
2	**protected int checkVerticalKey(int key, String message)** Verify that key is a legal value for the verticalAlignment or verticalTextPosition properties.

3	**AccessibleContext getAccessibleContext()** Get the AccessibleContext of this object.
4	**Icon getDisabledIcon()** Returns the icon used by the label when it's disabled.
5	**int getDisplayedMnemonic()** Return the keycode that indicates a mnemonic key.
6	**int getDisplayedMnemonicIndex()** Returns the character, as an index, that the look and feel should provide decoration for as representing the mnemonic character.
7	**int getHorizontalAlignment()** Returns the alignment of the label's contents along the **X** axis.
8	**int getHorizontalTextPosition()** Returns the horizontal position of the label's text, relative to its image.
9	**Icon getIcon()** Returns the graphic image (glyph, icon) that the label displays.
10	**int getIconTextGap()** Returns the amount of space between the text and the icon displayed in this label.
11	**Component getLabelFor()** Get the component this is labelling.
12	**String getText()** Returns the text string that the label displays.
13	**LabelUI getUI()** Returns the L&F object that renders this component.
14	**String getUIClassID()** Returns a string that specifies the name of the l&f class which renders this component.
15	

	int getVerticalAlignment() Returns the alignment of the label's contents along the **Y** axis.
16	**int getVerticalTextPosition()** Returns the vertical position of the label's text, relative to its image.
17	**boolean imageUpdate(Image img, int infoflags, int x, int y, int w, int h)** This is overridden to return false if the current Icon's image is not equal to the passed in Image img.
18	**protected String paramString()** Returns a string representation of this JLabel.
19	**void setDisabledIcon(Icon disabledIcon)** Sets the icon to be displayed if this JLabel is "disabled" (JLabel.setEnabled(false)).
20	**void setDisplayedMnemonic(char aChar)** Specifies the displayedMnemonic as a char value.
21	**void setDisplayedMnemonic(int key)** Specifies a keycode that indicates a mnemonic key.
22	**void setDisplayedMnemonicIndex(int index)** Provides a hint to the look and feel as to which character in the text should be decorated to represent the mnemonic.
23	**void setHorizontalAlignment(int alignment)** Sets the alignment of the label's contents along the **X** axis.
24	**void setHorizontalTextPosition(int textPosition)** Sets the horizontal position of the label's text, relative to its image.
25	**void setIcon(Icon icon)** Defines the icon this component will display.
26	**void setIconTextGap(int iconTextGap)**

	If both the icon and text properties are set, this property defines the space between them.
27	**void setLabelFor(Component c)** Sets the component, this is labelling.
28	**void setText(String text)** Defines the single line of text this component will display.
29	**void setUI(LabelUI ui)** Sets the L&F object that renders this component.
30	**void setVerticalAlignment(int alignment)** Sets the alignment of the label's contents along the **Y** axis.
31	**void setVerticalTextPosition(int textPosition)** Sets the vertical position of the label's text, relative to its image.
32	**void updateUI()** Resets the UI property to a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JLabel Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;


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

```
import javax.swing.*;

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
      swingControlDemo.showLabelDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
```

```
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showLabelDemo(){
      headerLabel.setText("Control in action: JLabel");

      JLabel label  = new JLabel("", JLabel.CENTER);
      label.setText("Welcome to TutorialsPoint Swing Tutorial.");
      label.setOpaque(true);
      label.setBackground(Color.GRAY);
      label.setForeground(Color.WHITE);
      controlPanel.add(label);

      mainFrame.setVisible(true);
   }
}
```

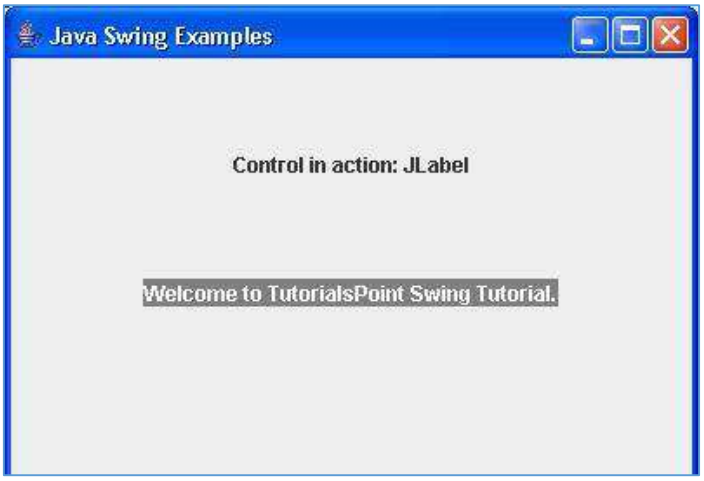
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.



JButton Class

Introduction

The class **JButton** is an implementation of a push button. This component has a label and generates an event when pressed. It can also have an Image.

Class Declaration

Following is the declaration for **javax.swing.JButton** class –

```
public class JButton
   extends AbstractButton
      implements Accessible
```

Class Constructors

Sr.No.	Constructor & Description
1	**JButton()** Creates a button with no set text or icon.
2	**JButton(Action a)** Creates a button where properties are taken from the Action supplied.
3	**JButton(Icon icon)** Creates a button with an icon.
4	**JButton(String text)** Creates a button with the text.
5	**JButton(String text, Icon icon)** Creates a button with an initial text and an icon.

Class Methods

Sr.No.	Method & Description
1	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JButton.
2	**String getUIClassID()** Returns a string that specifies the name of the L&F class which renders this component.
3	**boolean isDefaultButton()** Gets the value of the defaultButton property, which if true means that this button is the current default button for its JRootPane.
4	**boolean isDefaultCapable()** Gets the value of the defaultCapable property.
5	**protected String paramString()** Returns a string representation of this JButton.
6	**void removeNotify()** Overrides JComponent.removeNotify to check if this button is currently set as the default button on the RootPane. And if so, sets the RootPane's default button to null to ensure the RootPane doesn't hold onto an invalid button reference.
7	**void setDefaultCapable(boolean defaultCapable)** Sets the defaultCapable property, which determines whether this button can be made the default button for its root pane.
8	**void updateUI()** Resets the UI property to a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.AbstractButton
- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JButton Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
      swingControlDemo.showButtonDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
```

```
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private static ImageIcon createImageIcon(String path,
      String description) {
      java.net.URL imgURL = SwingControlDemo.class.getResource(path);
      if (imgURL != null) {
         return new ImageIcon(imgURL, description);
      } else {
         System.err.println("Couldn't find file: " + path);
         return null;
      }
   }
   private void showButtonDemo(){
      headerLabel.setText("Control in action: Button");

      //resources folder should be inside SWING folder.
      ImageIcon icon = createImageIcon("/resources/java_icon.png","Java");
```

```
      JButton okButton = new JButton("OK");
      JButton javaButton = new JButton("Submit", icon);
      JButton cancelButton = new JButton("Cancel", icon);
      cancelButton.setHorizontalTextPosition(SwingConstants.LEFT);

      okButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            statusLabel.setText("Ok Button clicked.");
         }
      });

      javaButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            statusLabel.setText("Submit Button clicked.");
         }
      });

      cancelButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            statusLabel.setText("Cancel Button clicked.");
         }
      });

      controlPanel.add(okButton);
      controlPanel.add(javaButton);
      controlPanel.add(cancelButton);

      mainFrame.setVisible(true);
   }
}
```

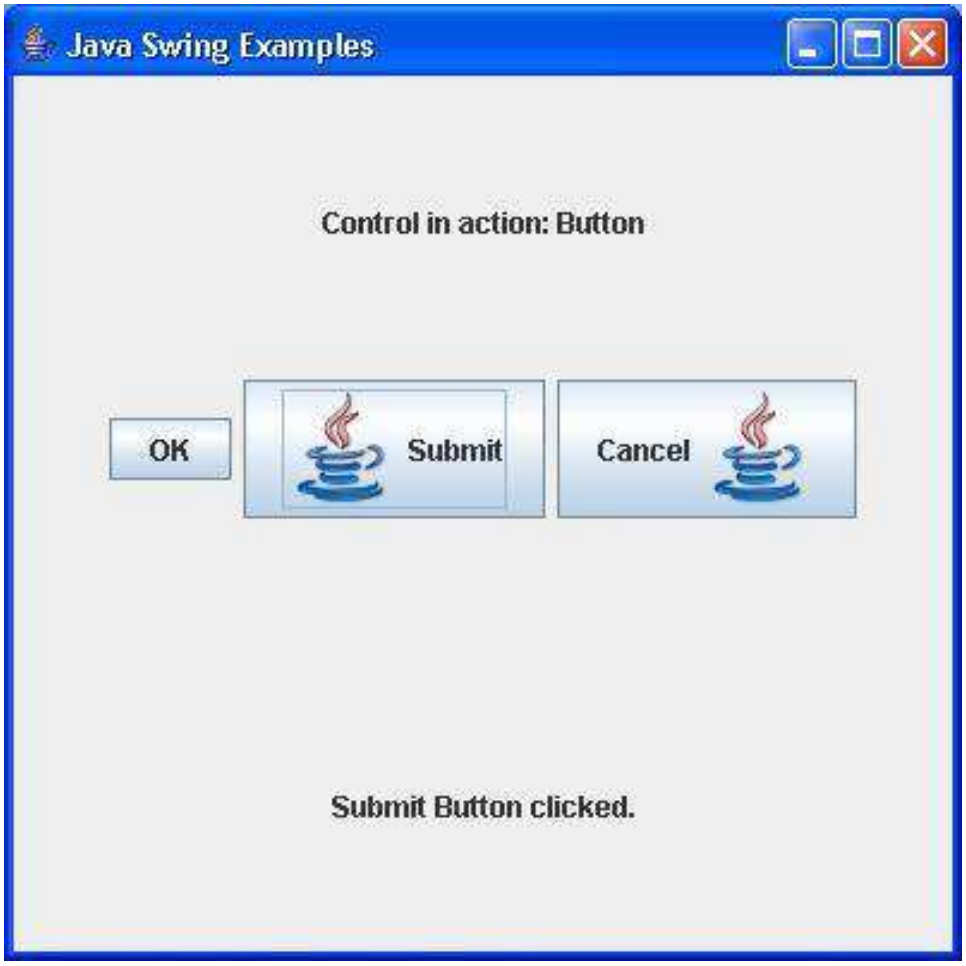
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, it means that the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.



JColorChooser Class

Introduction

The class **JColorChooser** provides a pane of controls designed to allow a user to manipulate and select a color.

Class Declaration

Following is the declaration for **javax.swing.JColorChooser** class –

```
public class JColorChooser
   extends JComponent
      implements Accessible
```

Field

Following are the fields for **javax.swing.JLabel** class –

- **protected AccessibleContext accessibleContext**
- **static String CHOOSER_PANELS_PROPERTY** – The chooserPanel array property name.
- **static String PREVIEW_PANEL_PROPERTY** – The preview panel property name.
- **static String SELECTION_MODEL_PROPERTY** – The selection model property name.

Class Constructors

Sr.No.	Constructor & Description
1	**JColorChooser()** Creates a color chooser pane with an initial color of white.
2	**JColorChooser(Color initialColor)** Creates a color chooser pane with the specified initial color.
3	**JColorChooser(ColorSelectionModel model)** Creates a color chooser pane with the specified ColorSelectionModel.

Class Methods

Sr.No.	Method & Description
1	**void addChooserPanel(AbstractColorChooserPanel panel)** Adds a color chooser panel to the color chooser.
2	**static JDialog createDialog(Component c, String title, boolean modal, JColorChooser chooserPane, ActionListener okListener, ActionListener cancelListener)** Creates and returns a new dialog containing the specified ColorChooser pane along with "OK", "Cancel", and "Reset" buttons.

3	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JColorChooser.
4	**AbstractColorChooserPanel[] getChooserPanels()** Returns the specified color panels.
5	**Color getColor()** Gets the current color value from the color chooser.
6	**boolean getDragEnabled()** Gets the value of the dragEnabled property.
7	**JComponent getPreviewPanel()** Returns the preview panel that shows a chosen color.
8	**ColorSelectionModel getSelectionModel()** Returns the data model that handles color selections.
9	**ColorChooserUI getUI()** Returns the L&F object that renders this component.
10	**String getUIClassID()** Returns the name of the L&F class that renders this component.
11	**protected String paramString()** Returns a string representation of this JColorChooser.
12	**AbstractColorChooserPanel removeChooserPanel(AbstractColorChooserPanel panel)** Removes the Color Panel specified.
13	**void setChooserPanels(AbstractColorChooserPanel[] panels)** Specifies the Color Panels used to choose a color value.
14	**void setColor(Color color)** Sets the current color of the color chooser to the specified color.
15	

	void setColor(int c) Sets the current color of the color chooser to the specified color.
16	**void setColor(int r, int g, int b)** Sets the current color of the color chooser to the specified RGB color.
17	**void setDragEnabled(boolean b)** Sets the dragEnabled property, which must be true to enable automatic drag handling (the first part of drag and drop) on this component.
18	**void setPreviewPanel(JComponent preview)** Sets the current preview panel.
19	**void setSelectionModel(ColorSelectionModel newModel)** Sets the model containing the selected color.
20	**void setUI(ColorChooserUI ui)** Sets the L&F object that renders this component.
21	**static Color showDialog(Component component, String title, Color initialColor)** Shows a modal color-chooser dialog and blocks until the dialog is hidden.
22	**void updateUI()** Notification from the UIManager that the L&F has changed.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JColorChooser Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
      swingControlDemo.showColorChooserDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
```

```
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showColorChooserDemo(){
      headerLabel.setText("Control in action: JColorChooser");

      JButton chooseButton = new JButton("Choose Background");
      chooseButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            Color backgroundColor = JColorChooser.showDialog(mainFrame,
               "Choose background color", Color.white);
            if(backgroundColor != null){
               controlPanel.setBackground(backgroundColor);
               mainFrame.getContentPane().setBackground(backgroundColor);
            }
         }
      });

      controlPanel.add(chooseButton);
      mainFrame.setVisible(true);
   }
}
```

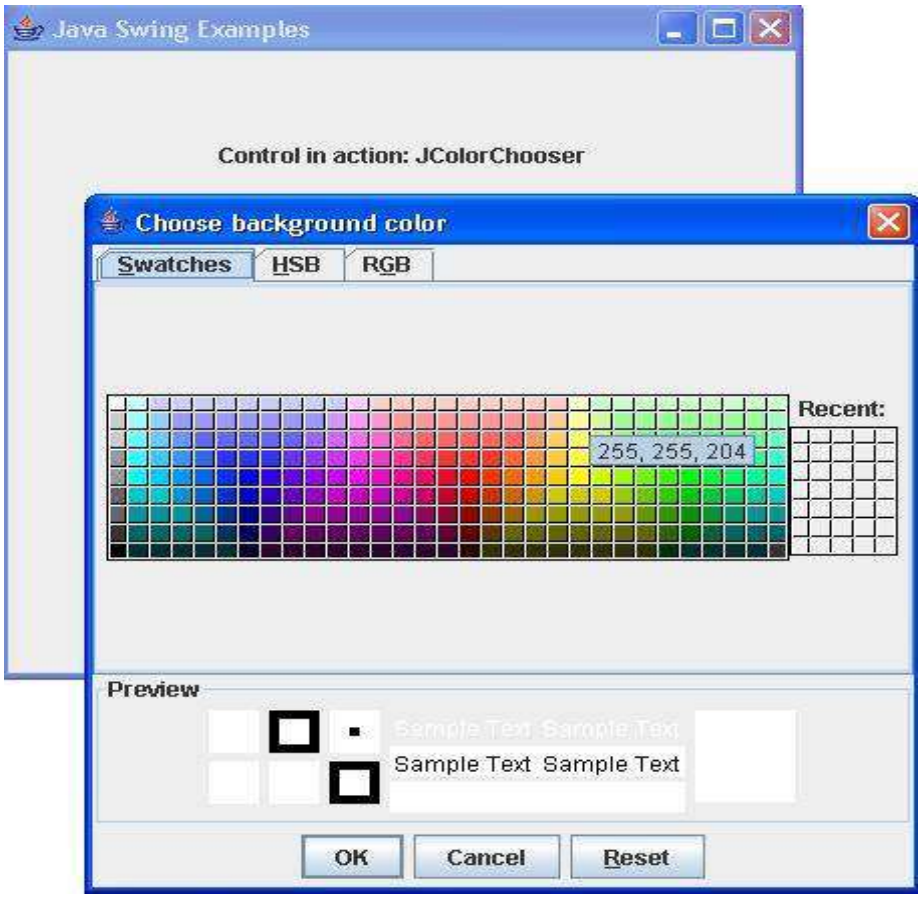
Compile the program using command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, then it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.



JCheckBox Class

Introduction

The class **JCheckBox** is an implementation of a checkbox - an item that can be selected or deselected, and which displays its state to the user.

Class Declaration

Following is the declaration for **javax.swing.JCheckBox** class –

```
public class JCheckBox
   extends JToggleButton
      implements Accessible
```

Field

Following are the fields for **javax.swing.JCheckBox** class –

- **static String BORDER_PAINTED_FLAT_CHANGED_PROPERTY** – Identifies a change to the flat property.

Class Constructors

Sr.No.	Constructor & Description
1	**JCheckBox()** Creates an initially unselected checkbox button with no text and no icon.
2	**JCheckBox(Action a)** Creates a checkbox where the properties are taken from the Action supplied.
3	**JCheckBox(Icon icon)** Creates an initially unselected checkbox with an icon.
4	**JCheckBox(Icon icon, boolean selected)** Creates a checkbox with an icon and specifies whether or not it is initially selected.
5	**JCheckBox(String text)** Creates an initially unselected checkbox with text.
6	**JCheckBox(String text, boolean selected)** Creates a checkbox with the text and specifies whether or not it is initially selected.
7	**JCheckBox(String text, Icon icon)** Creates an initially unselected checkbox with the specified text and icon.
8	**JCheckBox(String text, Icon icon, boolean selected)** Creates a checkbox with text and icon, and specifies whether or not it is initially selected.

Class Methods

Sr.No.	Method & Description
1	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JCheckBox.
2	**String getUIClassID()** Returns a string that specifies the name of the L&F class which renders this component.
3	**boolean isBorderPaintedFlat()** Gets the value of the borderPaintedFlat property.
4	**protected String paramString()** Returns a string representation of this JCheckBox.
5	**void setBorderPaintedFlat(boolean b)** Sets the borderPaintedFlat property, which gives a hint to the look and feel as to the appearance of the checkbox border.
6	**void updateUI()** Resets the UI property to a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.AbstractButton
- javax.swing.JToggleButton
- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JCheckBox Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;
```

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

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
      swingControlDemo.showCheckBoxDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());
```

```
      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showCheckBoxDemo(){

      headerLabel.setText("Control in action: CheckBox");

      final JCheckBox chkApple = new JCheckBox("Apple");
      final JCheckBox chkMango = new JCheckBox("Mango");
      final JCheckBox chkPeer = new JCheckBox("Peer");

      chkApple.setMnemonic(KeyEvent.VK_C);
      chkMango.setMnemonic(KeyEvent.VK_M);
      chkPeer.setMnemonic(KeyEvent.VK_P);

      chkApple.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
            statusLabel.setText("Apple Checkbox: "
            + (e.getStateChange()==1?"checked":"unchecked"));
         }
      });

      chkMango.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
            statusLabel.setText("Mango Checkbox: "
            + (e.getStateChange()==1?"checked":"unchecked"));
         }
      });

      chkPeer.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
            statusLabel.setText("Peer Checkbox: "
```

```
            + (e.getStateChange()==1?"checked":"unchecked"));
         }
      });

      controlPanel.add(chkApple);
      controlPanel.add(chkMango);
      controlPanel.add(chkPeer);

      mainFrame.setVisible(true);
   }
}
```

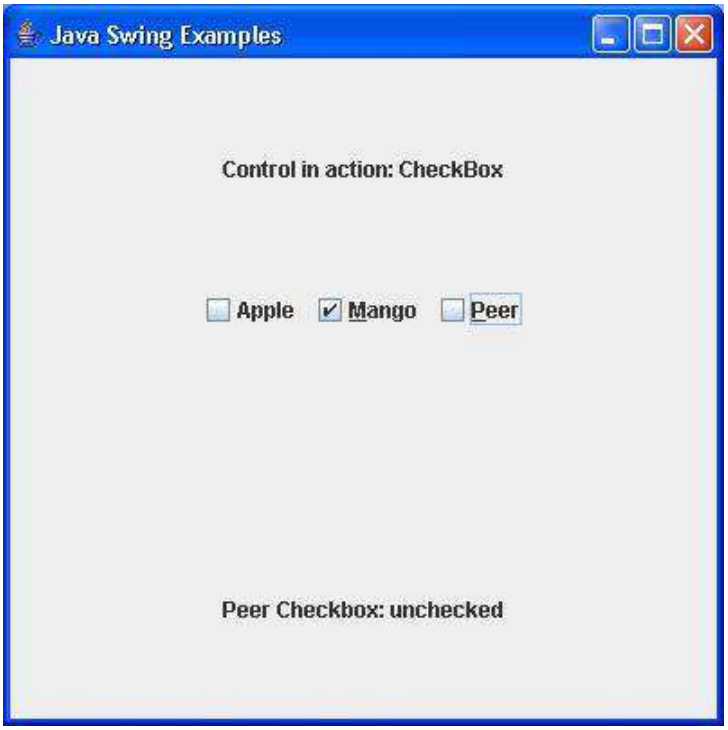
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, it mean the compilation is successful. Run the program using following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.



JRadioButton Class

Introduction

The class **JRadioButton** is an implementation of a radio button - an item that can be selected or deselected, and which displays its state to the user.

Class Declaration

Following is the declaration for **javax.swing.JRadioButton** class –

```
public class JRadioButton
   extends JToggleButton
      implements Accessible
```

Class Constructors

Sr.No.	Constructor & Description
1	**JRadioButton()** Creates an initially unselected radio button with no set text.
2	**JRadioButton(Action a)** Creates a radiobutton where properties are taken from the Action supplied.
3	**JRadioButton(Icon icon)** Creates an initially unselected radio button with the specified image but no text.
4	**JRadioButton(Icon icon, boolean selected)** Creates a radio button with the specified image and selection state, but no text.
5	**JRadioButton(String text, boolean selected)** Creates a radio button with the specified text and selection state.
6	**JRadioButton(String text, Icon icon)** Creates a radio button that has the specified text and image, and which is initially unselected.
7	**JRadioButton(String text, Icon icon, boolean selected)**

	Creates a radio button that has the specified text, image, and selection state.

Class Methods

Sr.No.	Method & Description
1	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JRadioButton.
2	**String getUIClassID()** Returns the name of the L&F class that renders this component.
3	**protected String paramString()** Returns a string representation of this JRadioButton.
4	**void updateUI()** Resets the UI property to a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.AbstractButton
- javax.swing.JToggleButton
- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JRadioButton Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
      swingControlDemo.showRadioButtonDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
```

```
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showRadioButtonDemo(){
      headerLabel.setText("Control in action: RadioButton");

      final JRadioButton radApple = new JRadioButton("Apple", true);
      final JRadioButton radMango = new JRadioButton("Mango");
      final JRadioButton radPeer = new JRadioButton("Peer");

      radApple.setMnemonic(KeyEvent.VK_C);
      radMango.setMnemonic(KeyEvent.VK_M);
      radPeer.setMnemonic(KeyEvent.VK_P);

      radApple.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
            statusLabel.setText("Apple RadioButton: "
            + (e.getStateChange()==1?"checked":"unchecked"));
         }
      });

      radMango.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
            statusLabel.setText("Mango RadioButton: "
            + (e.getStateChange()==1?"checked":"unchecked"));
         }
      });

      radPeer.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
```

```
            statusLabel.setText("Peer RadioButton: "
            + (e.getStateChange()==1?"checked":"unchecked"));
         }
      });

      //Group the radio buttons.
      ButtonGroup group = new ButtonGroup();
      group.add(radApple);
      group.add(radMango);
      group.add(radPeer);

      controlPanel.add(radApple);
      controlPanel.add(radMango);
      controlPanel.add(radPeer);

      mainFrame.setVisible(true);
   }
}
```

Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.

JList Class

Introduction

The class **JList** is a component which displays a list of objects and allows the user to select one or more items. A separate model, ListModel, maintains the contents of the list.

Class Declaration

Following is the declaration for **javax.swing.JList** class:

```
public class JList
   extends JComponent
      implements Scrollable, Accessible
```

Field

Following are the fields for **javax.swing.JList** class –

- **static int HORIZONTAL_WRAP** – Indicates a "newspaper style" layout with cells flowing horizontally then vertically.
- **static int VERTICAL** – Indicates a vertical layout of cells, in a single column; the default layout.
- **static int VERTICAL_WRAP** – Indicates a "newspaper style" layout with cells flowing vertically then horizontally.

Class Constructors

Sr.No.	Constructor & Description
1	**JList()** Constructs a JList with an empty, read-only, model.
2	**JList(ListModel dataModel)** Constructs a JList that displays elements from the specified, non-null, model.
3	**JList(Object[] listData)** Constructs a JList that displays the elements in the specified array.
4	**JList(Vector<?> listData)** Constructs a JList that displays the elements in the specified vector.

Class Methods

Sr.No.	Method & Description
1	**void addListSelectionListener(ListSelectionListener listener)** Adds a listener to the list, to be notified each time a change to the selection occurs; the preferred way of listening for selection state changes.
2	**void addSelectionInterval(int anchor, int lead)** Sets the selection to be the union of the specified interval with the current selection.
3	**void clearSelection()** Clears the selection; after calling this method, isSelectionEmpty will return true.
4	**protected ListSelectionModel createSelectionModel()** Returns an instance of DefaultListSelectionModel; called during construction to initialize the list's selection model property.

5	**void ensureIndexIsVisible(int index)** Scrolls the list within an enclosing viewport to make the specified cell completely visible.
6	**protected void fireSelectionValueChanged(int firstIndex, int lastIndex, boolean isAdjusting)** Notifies ListSelectionListeners added directly to the list of selection changes made to the selection model.
7	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JList.
8	**int getAnchorSelectionIndex()** Returns the anchor selection index.
9	**Rectangle getCellBounds(int index0, int index1)** Returns the bounding rectangle, in the list's coordinate system, for the range of cells specified by the two indices.
10	**ListCellRenderer getCellRenderer()** Returns the object responsible for painting list items.
11	**boolean getDragEnabled()** Returns whether or not automatic drag handling is enabled.
12	**JList.DropLocation getDropLocation()** Returns the location that this component should visually indicate as the drop location, during a DnD operation over the component, or null if no location is to currently be shown.
13	**DropMode getDropMode()** Returns the drop mode for this component.
14	**int getFirstVisibleIndex()** Returns the smallest list index that is currently visible.
15	**int getFixedCellHeight()**

	Returns the value of the fixedCellHeight property.
16	**int getFixedCellWidth()** Returns the value of the fixedCellWidth property.
17	**int getLastVisibleIndex()** Returns the largest list index that is currently visible.
18	**int getLayoutOrientation()** Returns the layout orientation property for the list: VERTICAL if the layout is a single column of cells, VERTICAL_WRAP if the layout is "newspaper style" with the content flowing vertically then horizontally, or HORIZONTAL_WRAP if the layout is "newspaper style" with the content flowing horizontally then vertically.
19	**int getLeadSelectionIndex()** Returns the lead selection index.
20	**ListSelectionListener[] getListSelectionListeners()** Returns an array of all the ListSelectionListeners added to this JList by way of addListSelectionListener.
21	**int getMaxSelectionIndex()** Returns the largest selected cell index, or -1 if the selection is empty.
22	**int getMinSelectionIndex()** Returns the smallest selected cell index, or -1 if the selection is empty.
23	**ListModel getModel()** Returns the data model that holds the list of items displayed by the JList component.
24	**int getNextMatch(String prefix, int startIndex, Position.Bias bias)** Returns the next list element whose toString value starts with the given prefix.
25	**Dimension getPreferredScrollableViewportSize()** Computes the size of viewport needed to display visibleRowCount rows.
26	

	Object getPrototypeCellValue() Returns the "prototypical" cell value - a value used to calculate a fixed width and height for cells.
27	**int getScrollableBlockIncrement(Rectangle visibleRect, int orientation, int direction)** Returns the distance to scroll to expose the next or previous block.
28	**boolean getScrollableTracksViewportHeight()** Returns true if this JList is displayed in a JViewport and the viewport is taller than the list's preferred height, or if the layout orientation is VERTICAL_WRAP and visibleRowCount <= 0; otherwise returns false.
29	**boolean getScrollableTracksViewportWidth()** Returns true if this JList is displayed in a JViewport and the viewport is wider than the list's preferred width, or if the layout orientation is HORIZONTAL_WRAP and visibleRowCount <= 0; otherwise returns false.
30	**int getScrollableUnitIncrement(Rectangle visibleRect, int orientation, int direction)** Returns the distance to scroll to expose the next or previous row (for vertical scrolling) or column (for horizontal scrolling).
31	**int getSelectedIndex()** Returns the smallest selected cell index; the selection when only a single item is selected in the list.
32	**int[] getSelectedIndices()** Returns an array of all of the selected indices, in an increasing order.
33	**Object getSelectedValue()** Returns the value for the smallest selected cell index; the selected value when only a single item is selected in the list.
34	**Object[] getSelectedValues()** Returns an array of all the selected values, in an increasing order based on their indices in the list.
35	**Color getSelectionBackground()**

	Returns the color used to draw the background of selected items.
36	**Color getSelectionForeground()** Returns the color used to draw the foreground of selected items.
37	**int getSelectionMode()** Returns the current selection mode for the list.
38	**ListSelectionModel getSelectionModel()** Returns the current selection model.
39	**String getToolTipText(MouseEvent event)** Returns the tooltip text to be used for the given event.
40	**ListUI getUI()** Returns the ListUI, the look and feel object that renders this component.
41	**String getUIClassID()** Returns "ListUI", the UIDefaults key used to look up the name of the javax.swing.plaf.ListUI class that defines the look and feel for this component.
42	**boolean getValueIsAdjusting()** Returns the value of the selection model's isAdjusting property.
43	**int getVisibleRowCount()** Returns the value of the visibleRowCount property.
44	**Point indexToLocation(int index)** Returns the origin of the specified item in the list's coordinate system.
45	**boolean isSelectedIndex(int index)** Returns true, if the specified index is selected, else false.
46	**boolean isSelectionEmpty()** Returns true if nothing is selected, else false.
47	**int locationToIndex(Point location)**

	Returns the cell index closest to the given location in the list's coordinate system.
48	**protected String paramString()** Returns a String representation of this JList.
49	**void removeListSelectionListener(ListSelectionListener listener)** Removes a selection listener from the list.
50	**void removeSelectionInterval(int index0, int index1)** Sets the selection to be the set difference of the specified interval and the current selection.
51	**void setCellRenderer(ListCellRenderer cellRenderer)** Sets the delegate that is used to paint each cell in the list.
52	**void setDragEnabled(boolean b)** Turns the automatic drag handling on or off.
53	**void setDropMode(DropMode dropMode)** Sets the drop mode for this component.
54	**void setFixedCellHeight(int height)** Sets a fixed value to be used for the height of every cell in the list.
55	**void setFixedCellWidth(int width)** Sets a fixed value to be used for the width of every cell in the list.
56	**void setLayoutOrientation(int layoutOrientation)** Defines the way list cells are layed out.
57	**void setListData(Object[] listData)** Constructs a read-only ListModel from an array of objects, and calls setModel with this model.
58	**void setListData(Vector<?> listData)** Constructs a read-only ListModel from a Vector and calls setModel with this model.

59	**void setModel(ListModel model)** Sets the model that represents the contents or "value" of the list, notifies property change listeners, and then clears the list's selection.
60	**void setPrototypeCellValue(Object prototypeCellValue)** Sets the prototypeCellValue property, and then (if the new value is non-null), computes the fixedCellWidth and fixedCellHeight properties by requesting the cell renderer component for the given value (and index 0) from the cell renderer, and using that component's preferred size.
61	**void setSelectedIndex(int index)** Selects a single cell.
62	**void setSelectedIndices(int[] indices)** Changes the selection to be the set of indices specified by the given array.
63	**void setSelectedValue(Object anObject, boolean shouldScroll)** Selects the specified object from the list.
64	**void setSelectionBackground(Color selectionBackground)** Sets the color used to draw the background of the selected items, which cell renderers can use to fill the selected cells.
65	**void setSelectionForeground(Color selectionForeground)** Sets the color used to draw the foreground of the selected items, which cell renderers can use to render text and graphics.
66	**void setSelectionInterval(int anchor, int lead)** Selects the specified interval.
67	**void setSelectionMode(int selectionMode)** Sets the selection mode for the list.
68	**void setSelectionModel(ListSelectionModel selectionModel)** Sets the selectionModel for the list to a non-null ListSelectionModel implementation.
69	

	void setUI(ListUI ui) Sets the ListUI, the look and feel object that renders this component.
70	**void setValueIsAdjusting(boolean b)** Sets the selection model's valueIsAdjusting property.
71	**void setVisibleRowCount(int visibleRowCount)** Sets the visibleRowCount property, which has different meanings depending on the layout orientation: For a VERTICAL layout orientation, this sets the preferred number of rows to display without requiring scrolling; for other orientations, it affects the wrapping of cells.
72	**void updateUI()** Resets the ListUI property by setting it to the value provided by the current look and feel.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JList Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
```

```
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
      swingControlDemo.showListDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showListDemo(){
```

```
      headerLabel.setText("Control in action: JList");

      final DefaultListModel fruitsName = new DefaultListModel();

      fruitsName.addElement("Apple");
      fruitsName.addElement("Grapes");
      fruitsName.addElement("Mango");
      fruitsName.addElement("Peer");

      final JList fruitList = new JList(fruitsName);
      fruitList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
      fruitList.setSelectedIndex(0);
      fruitList.setVisibleRowCount(3);

      JScrollPane fruitListScrollPane = new JScrollPane(fruitList);

      final DefaultListModel vegName = new DefaultListModel();

      vegName.addElement("Lady Finger");
      vegName.addElement("Onion");
      vegName.addElement("Potato");
      vegName.addElement("Tomato");

      final JList vegList = new JList(vegName);
      vegList.setSelectionMode(
         ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
      vegList.setSelectedIndex(0);
      vegList.setVisibleRowCount(3);

      JScrollPane vegListScrollPane = new JScrollPane(vegList);

      JButton showButton = new JButton("Show");

      showButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
```

```
            String data = "";
            if (fruitList.getSelectedIndex() != -1) {
               data = "Fruits Selected: " + fruitList.getSelectedValue();
               statusLabel.setText(data);
            }
            if(vegList.getSelectedIndex() != -1){
               data += " Vegetables selected: ";
               for(Object vegetable:vegList.getSelectedValues()){
                  data += vegetable + " ";
               }
            }
            statusLabel.setText(data);
         }
      });

      controlPanel.add(fruitListScrollPane);
      controlPanel.add(vegListScrollPane);
      controlPanel.add(showButton);

      mainFrame.setVisible(true);
   }
}
```

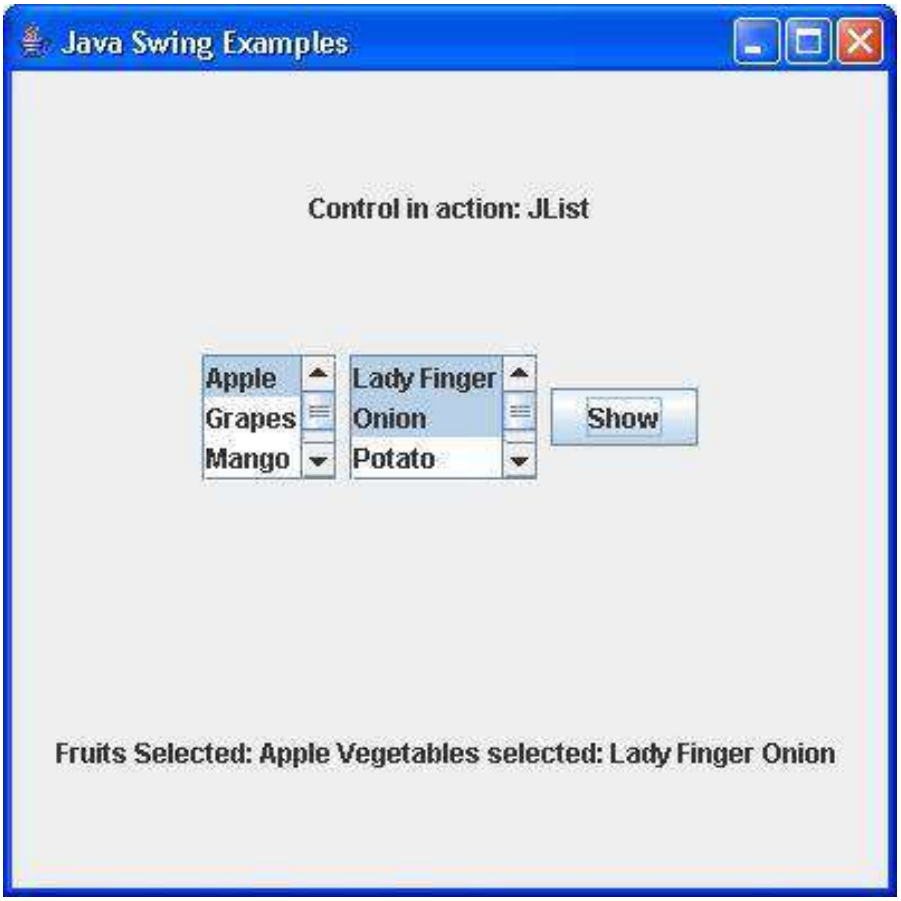
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.



JComboBox Class

Introduction

The class **JComboBox** is a component which combines a button or editable field and a drop-down list.

Class Declaration

Following is the declaration for **javax.swing.JComboBox** class –

```
public class JComboBox
   extends JComponent
      implements ItemSelectable, ListDataListener,
         ActionListener, Accessible
```

Field

Following are the fields for **javax.swing.JList** class –

- **protected String actionCommand** – This protected field is implementation specific.
- **protected ComboBoxModel dataModel** – This protected field is implementation specific.
- **protected ComboBoxEditor editor** – This protected field is implementation specific.
- **protected boolean isEditable** – This protected field is implementation specific.
- **protected JComboBox.KeySelectionManager keySelectionManager** – This protected field is implementation specific.
- **protected boolean lightWeightPopupEnabled** – This protected field is implementation specific.
- **protected int maximumRowCount** – This protected field is implementation specific.
- **protected ListCellRenderer renderer** – This protected field is implementation specific.
- **protected Object selectedItemReminder** – This protected field is implementation specific.

Class Constructors

Sr.No.	Constructor & Description
1	**JComboBox()** Creates a JComboBox with a default data model.
2	**JComboBox(ComboBoxModel aModel)** Creates a JComboBox that takes its items from an existing ComboBoxModel.
3	**JComboBox(Object[] items)** Creates a JComboBox that contains the elements in the specified array.
4	**JComboBox(Vector<?> items)** Creates a JComboBox that contains the elements in the specified Vector.

Class Methods

Sr.No.	Method & Description
1	**void actionPerformed(ActionEvent e)** This method is public as an implementation side effect.
2	**protected void actionPropertyChanged(Action action, String propertyName)** Updates the ComboBox's state in response to property changes in associated action.
3	**void addActionListener(ActionListener l)** Adds an ActionListener.
4	**void addItem(Object anObject)** Adds an item to the item list.
5	**void addItemListener(ItemListener aListener)** Adds an ItemListener.
6	**void addPopupMenuListener(PopupMenuListener l)** Adds a PopupMenu listener which will listen to notification messages from the popup portion of the ComboBox.
7	**void configureEditor(ComboBoxEditor anEditor, Object anItem)** Initializes the editor with the specified item.
8	**protected void configurePropertiesFromAction(Action a)** Sets the properties on this ComboBox to match those in the specified Action.
9	**void contentsChanged(ListDataEvent e)** This method is public as an implementation side effect.
10	**protected PropertyChangeListener createActionPropertyChangeListener(Action a)** Creates and returns a PropertyChangeListener responsible for listening to changes from the specified Action and updating the appropriate properties.
11	**protected JComboBox.KeySelectionManager createDefaultKeySelectionManager()** Returns an instance of the default key-selection manager.
12	**protected void fireActionEvent()** Notifies all listeners who have registered interest for notification on this event type.

13	**protected void fireItemStateChanged(ItemEvent e)** Notifies all listeners who have registered interest for notification on this event type.
14	**void firePopupMenuCanceled()** Notifies PopupMenuListeners that the popup portion of the ComboBox has been canceled.
15	**void firePopupMenuWillBecomeInvisible()** Notifies PopupMenuListeners that the popup portion of the ComboBox has become invisible.
16	**void firePopupMenuWillBecomeVisible()** Notifies PopupMenuListeners that the popup portion of the ComboBox will become visible.
17	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JComboBox.
18	**Action getAction()** Returns the currently set Action for this ActionEvent source, or null if no Action is set.
19	**String getActionCommand()** Returns the action command that is included in the event sent to action listeners.
20	**ActionListener[] getActionListeners()** Returns an array of all the ActionListeners added to this JComboBox with addActionListener().
21	**ComboBoxEditor getEditor()** Returns the editor used to paint and edit the selected item in the JComboBox field.
22	**Object getItemAt(int index)**

	Returns the list item at the specified index.
23	**int getItemCount()** Returns the number of items in the list.
24	**ItemListener[] getItemListeners()** Returns an array of all the ItemListeners added to this JComboBox with addItemListener().
25	**JComboBox.KeySelectionManager getKeySelectionManager()** Returns the list's key-selection manager.
26	**int getMaximumRowCount()** Returns the maximum number of items the ComboBox can display without a scrollbar.
27	**ComboBoxModel getModel()** Returns the data model currently used by the JComboBox.
28	**PopupMenuListener[]getPopupMenuListeners()** Returns an array of all the PopupMenuListeners added to this JComboBox with addPopupMenuListener().
29	**Object getPrototypeDisplayValue()** Returns the "prototypical display" value - an Object used for the calculation of the display height and width.
30	**ListCellRenderer getRenderer()** Returns the renderer used to display the selected item in the JComboBox field.
31	**int getSelectedIndex()** Returns the first item in the list that matches the given item.
32	**Object getSelectedItem()** Returns the current selected item.
33	**Object[] getSelectedObjects()** Returns an array containing the selected item.
34	**ComboBoxUI getUI()**

	Returns the L&F object that renders this component.
35	**String getUIClassID()** Returns the name of the L&F class that renders this component.
36	**void hidePopup()** Causes the ComboBox to close its popup window.
37	**void insertItemAt(Object anObject, int index)** Inserts an item into the item list at a given index.
38	**protected void installAncestorListener()**
39	**void intervalAdded(ListDataEvent e)** This method is public as an implementation side effect.
40	**void intervalRemoved(ListDataEvent e)** This method is public as an implementation side effect.
41	**boolean isEditable()** Returns true, if the JComboBox is editable.
42	**boolean isLightWeightPopupEnabled()** Gets the value of the lightWeightPopupEnabled property.
43	**boolean isPopupVisible()** Determines the visibility of the popup.
44	**protected String paramString()** Returns a string representation of this JComboBox.
45	**void processKeyEvent(KeyEvent e)** Handles KeyEvents, looking for the Tab key.
46	**void removeActionListener(ActionListener l)** Removes an ActionListener.
47	**void removeAllItems()** Removes all items from the item list.
48	**void removeItem(Object anObject)**

	Removes an item from the item list.
49	**void removeItemAt(int anIndex)** Removes the item at anIndex. This method works only if the JComboBox uses a mutable data model.
50	**void removeItemListener(ItemListener aListener)** Removes an ItemListener.
51	**void removePopupMenuListener(PopupMenuListener l)** Removes a PopupMenuListener.
52	**protected void selectedItemChanged()** This protected method is implementation specific.
53	**boolean selectWithKeyChar(char keyChar)** Selects the list item that corresponds to the specified keyboard character and returns true, if there is an item corresponding to that character.
54	**void setAction(Action a)** Sets the Action for the ActionEvent source.
55	**void setActionCommand(String aCommand)** Sets the action command that should be included in the event sent to action listeners.
56	**void setEditable(boolean aFlag)** Determines whether the JComboBox field is editable.
57	**void setEditor(ComboBoxEditor anEditor)** Sets the editor used to paint and edit the selected item in the JComboBox field.
58	**void setEnabled(boolean b)** Enables the ComboBox so that items can be selected.
59	**void setKeySelectionManager(JComboBox.KeySelectionManager aManager)**

	Sets the object that translates a keyboard character into a list selection.
60	**void setLightWeightPopupEnabled(boolean aFlag)** Sets the lightWeightPopupEnabled property, which provides a hint as to whether or not a lightweight Component should be used to contain the JComboBox, versus a heavyweight Component such as a Panel or a Window.
61	**void setMaximumRowCount(int count)** Sets the maximum number of rows the JComboBox displays.
62	**void setModel(ComboBoxModel aModel)** Sets the data model the JComboBox uses to obtain the list of items.
63	**void setPopupVisible(boolean v)** Sets the visibility of the popup.
64	**void setPrototypeDisplayValue(Object prototypeDisplayValue)** Sets the prototype display value used to calculate the size of the display for the UI portion.
65	**void setRenderer(ListCellRenderer aRenderer)** Sets the renderer that paints the list items and the item selected from the list in the JComboBox field.
66	**void setSelectedIndex(int anIndex)** Selects the item at index anIndex.
67	**void setSelectedItem(Object anObject)** Sets the selected item in the ComboBox display area to the object in the argument.
68	**void setUI(ComboBoxUI ui)** Sets the L&F object that renders this component.
69	**void showPopup()** Causes the ComboBox to display its popup window.
70	**void updateUI()**

	Resets the UI property to a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JComboBox Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
      swingControlDemo.showComboboxDemo();
   }

   private void prepareGUI(){
```

```
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showComboboxDemo(){
      headerLabel.setText("Control in action: JComboBox");

      final DefaultComboBoxModel fruitsName = new DefaultComboBoxModel();

      fruitsName.addElement("Apple");
      fruitsName.addElement("Grapes");
      fruitsName.addElement("Mango");
      fruitsName.addElement("Peer");

      final JComboBox fruitCombo = new JComboBox(fruitsName);
      fruitCombo.setSelectedIndex(0);
      JScrollPane fruitListScrollPane = new JScrollPane(fruitCombo);
```

```
      JButton showButton = new JButton("Show");

      showButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            String data = "";
            if (fruitCombo.getSelectedIndex() != -1) {
               data = "Fruits Selected: "
                  + fruitCombo.getItemAt
                    (fruitCombo.getSelectedIndex());
            }
            statusLabel.setText(data);
         }
      });
      controlPanel.add(fruitListScrollPane);
      controlPanel.add(showButton);
      mainFrame.setVisible(true);
   }
}
```

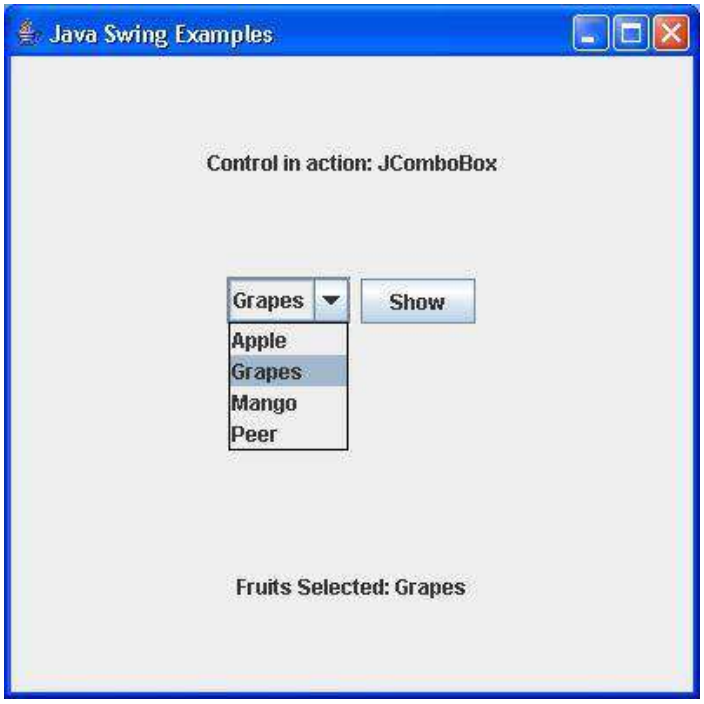
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.



JTextField Class

Introduction

The class **JTextField** is a component which allows editing of a single line of text.

Class Declaration

Following is the declaration for **javax.swing.JTextField** class –

```
public class JTextField
   extends JTextComponent
      implements SwingConstants
```

Field

Following are the fields for **javax.swing.JList** class –

- **static String notifyAction** – Name of the action to send notification that the contents of the field have been accepted.

Class Constructors

Sr.No.	Constructor & Description
1	**JTextField()** Constructs a new TextField.
2	**JTextField(Document doc, String text, int columns)** Constructs a new JTextField that uses the given text storage model and the given number of columns.
3	**JTextField(int columns)** Constructs a new empty TextField with the specified number of columns.
4	**JTextField(String text)** Constructs a new TextField initialized with the specified text.
5	**JTextField(String text, int columns)** Constructs a new TextField initialized with the specified text and columns.

Class Methods

Sr.No.	Method & Description
1	**protected void actionPropertyChanged(Action action, String propertyName)** Updates the textfield's state in response to property changes in associated action.
2	**void addActionListener(ActionListener l)** Adds the specified action listener to receive action events from this textfield.

3	**protected void configurePropertiesFromAction(Action a)** Sets the properties on this textfield to match those in the specified Action.
4	**protected PropertyChangeListener createActionPropertyChangeListener(Action a)** Creates and returns a PropertyChangeListener responsible for listening to changes from the specified Action and updating the appropriate properties.
5	**protected Document createDefaultModel()** Creates the default implementation of the model to be used at construction, if one isn't explicitly given.
6	**protected void fireActionPerformed()** Notifies all listeners who have registered interest for notification on this event type.
7	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JTextField.
8	**Action getAction()** Returns the currently set Action for this ActionEvent source, or null if no Action is set.
9	**ActionListener[] getActionListeners()** Returns an array of all the ActionListeners added to this JTextField with addActionListener().
10	**Action[] getActions()** Fetches the command list for the editor.
11	**int getColumns()** Returns the number of columns in this TextField.
12	**protected int getColumnWidth()** Returns the column width.

13	**int getHorizontalAlignment()** Returns the horizontal alignment of the text.
14	**BoundedRangeModel getHorizontalVisibility()** Gets the visibility of the text field.
15	**Dimension getPreferredSize()** Returns the preferred size Dimensions needed for this TextField.
16	**int getScrollOffset()** Gets the scroll offset, in pixels.
17	**String getUIClassID()** Gets the class ID for a UI.
18	**boolean isValidateRoot()** Calls to revalidate that come from within the textfield itself will be handled by validating the textfield, unless the textfield is contained within a JViewport, in which case this returns false.
19	**protected String paramString()** Returns a string representation of this JTextField.
20	**void postActionEvent()** Processes action events occurring on this textfield by dispatching them to any registered ActionListener objects.
21	**void removeActionListener(ActionListener l)** Removes the specified action listener so that it no longer receives action events from this textfield.
22	**void scrollRectToVisible(Rectangle r)** Scrolls the field left or right.
23	**void setAction(Action a)** Sets the Action for the ActionEvent source.

24	**void setActionCommand(String command)** Sets the command string used for action events.
25	**void setColumns(int columns)** Sets the number of columns in this TextField, and then invalidate the layout.
26	**void setDocument(Document doc)** Associates the editor with a text document.
27	**void setFont(Font f)** Sets the current font.
28	**void setHorizontalAlignment(int alignment)** Sets the horizontal alignment of the text.
29	**void setScrollOffset(int scrollOffset)** Sets the scroll offset, in pixels.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.text.JTextComponent
- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JTextField Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

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

```
public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
      swingControlDemo.showTextFieldDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
```

```
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showTextFieldDemo(){
      headerLabel.setText("Control in action: JTextField");

      JLabel  namelabel= new JLabel("User ID: ", JLabel.RIGHT);
      JLabel  passwordLabel = new JLabel("Password: ", JLabel.CENTER);
      final JTextField userText = new JTextField(6);
      final JPasswordField passwordText = new JPasswordField(6);

      JButton loginButton = new JButton("Login");
      loginButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            String data = "Username " + userText.getText();
            data += ", Password: "
            + new String(passwordText.getPassword());
            statusLabel.setText(data);
         }
      });

      controlPanel.add(namelabel);
      controlPanel.add(userText);
      controlPanel.add(passwordLabel);
      controlPanel.add(passwordText);
      controlPanel.add(loginButton);
      mainFrame.setVisible(true);
   }
}
```

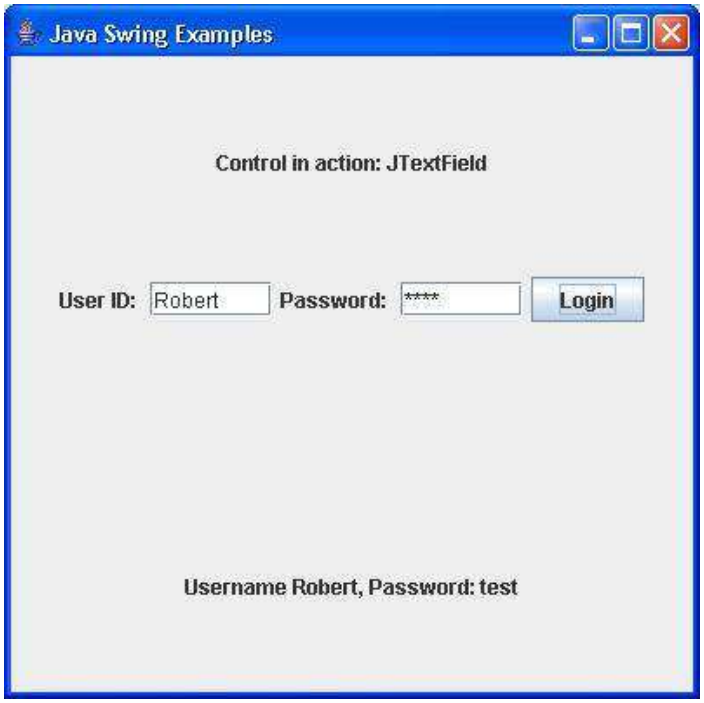
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.



JPassword Field Class

Introduction

The class **JPasswordField** is a component which is specialized to handle password functionality and allows the editing of a single line of text.

Class declaration

Following is the declaration for **javax.swing.JPasswordField** class:

```
public class JPasswordField
   extends JTextField
```

Class constructors

S.N.	Constructor & Description
1	**JPasswordField()** Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.

2	**JPasswordField(Document doc, String txt, int columns)** Constructs a new JPasswordField that uses the given text storage model and the given number of columns.
3	**JPasswordField(int columns)** Constructs a new empty JPasswordField with the specified number of columns.
4	**JPasswordField(String text)** Constructs a new JPasswordField initialized with the specified text.
5	**JPasswordField(String text, int columns)** Constructs a new JPasswordField initialized with the specified text and columns.

Class methods

S.N.	Method & Description
1	**void copy()** Invokes provideErrorFeedback on the current look and feel, which typically initiates an error beep.
2	**void cut()** Invokes provideErrorFeedback on the current look and feel, which typically initiates an error beep.
3	**boolean echoCharIsSet()** Returns true if this JPasswordField has a character set for echoing.
4	**AccessibleContext getAccessibleContext()** Returns the AccessibleContext associated with this JPasswordField.
5	**char getEchoChar()** Returns the character to be used for echoing.
6	**char[] getPassword()** Returns the text contained in this TextComponent.
7	**String getText()** Deprecated. As of Java 2 platform v1.2, replaced by getPassword.
8	**String getText(int offs, int len)** Deprecated. As of Java 2 platform v1.2, replaced by getPassword.
9	**String getUIClassID()** Returns the name of the L&F class that renders this component.
10	**protected String paramString()** Returns a string representation of this JPasswordField.

11	**void setEchoChar(char c)** Sets the echo character for this JPasswordField.
12	**void updateUI()** Reloads the pluggable UI.

Methods inherited

This class inherits methods from the following classes:

- javax.swing.JTextField
- javax.swing.text.JTextComponent
- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JPasswordField Example

Create the following java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
```

```
      swingControlDemo.showPasswordFieldDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showPasswordFieldDemo(){
      headerLabel.setText("Control in action: JPasswordField");

      JLabel  namelabel= new JLabel("User ID: ", JLabel.RIGHT);
      JLabel  passwordLabel = new JLabel("Password: ", JLabel.CENTER);
      final JTextField userText = new JTextField(6);
      final JPasswordField passwordText = new JPasswordField(6);
      passwordText.setEchoChar('~');

      JButton loginButton = new JButton("Login");
      loginButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
```

```
            String data = "Username " + userText.getText();
            data += ", Password: "
            + new String(passwordText.getPassword());
            statusLabel.setText(data);
         }
      });

      controlPanel.add(namelabel);
      controlPanel.add(userText);
      controlPanel.add(passwordLabel);
      controlPanel.add(passwordText);
      controlPanel.add(loginButton);
      mainFrame.setVisible(true);
   }
}
```

Compile the program using command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error comes that means compilation is successful. Run the program using following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.

JTextArea Class

Introduction

The class **JTextArea** is a multi-line area to display plain text.

Class Declaration

Following is the declaration for **javax.swing.JTextArea** class –

```
public class JTextArea
   extends JTextComponent
```

Class Constructors

Sr.No.	Constructor & Description
1	**JTextArea()** Constructs a new TextArea.
2	**JTextArea(Document doc)** Constructs a new JTextArea with the given document model, and defaults for all of the other arguments (null, 0, 0).
3	**JTextArea(Document doc, String text, int rows, int columns)** Constructs a new JTextArea with the specified number of rows and columns, and the given model.
4	**JTextArea(int rows, int columns)** Constructs a new empty TextArea with the specified number of rows and columns.
5	**JTextArea(String text)** Constructs a new TextArea with the specified text displayed.
6	**JTextArea(String text, int rows, int columns)** Constructs a new TextArea with the specified text, and number of rows and columns.

Class Methods

Sr.No.	Method & Description
1	**void append(String str)** Appends the given text to the end of the document.
2	**protected Document createDefaultModel()** Creates the default implementation of the model to be used at construction, if one isn't explicitly given.
3	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JTextArea.
4	**int getColumns()** Returns the number of columns in the TextArea.
5	**protected int getColumnWidth()** Gets column width.
6	**int getLineCount()** Determines the number of lines contained in the area.
7	**int getLineEndOffset(int line)** Determines the offset of the end of the given line.
8	**int getLineOfOffset(int offset)** Translates an offset into the components text to a line number.
9	**int getLineStartOffset(int line)** Determines the offset of the start of the given line.
10	**boolean getLineWrap()** Gets the line-wrapping policy of the text area.
11	**Dimension getPreferredScrollableViewportSize()**

	Returns the preferred size of the viewport, if this component is embedded in a JScrollPane.
12	**Dimension getPreferredSize()** Returns the preferred size of the TextArea.
13	**protected int getRowHeight()** Defines the meaning of the height of a row.
14	**int getRows()** Returns the number of rows in the TextArea.
15	**boolean getScrollableTracksViewportWidth()** Returns true if a viewport should always force the width of this Scrollable to match the width of the viewport.
16	**int getScrollableUnitIncrement(Rectangle visibleRect, int orientation, int direction)** Components that display logical rows or columns should compute the scroll increment that will completely expose one new row or column, depending on the value of orientation.
17	**int getTabSize()** Gets the number of characters used to expand tabs.
18	**String getUIClassID()** Returns the class ID for the UI.
19	**boolean getWrapStyleWord()** Gets the style of wrapping used if the text area is wrapping lines.
20	**void insert(String str, int pos)** Inserts the specified text at the specified position.
21	**protected String paramString()** Returns a string representation of this JTextArea.
22	**void replaceRange(String str, int start, int end)**

	Replaces text from the indicated start to end position with the new text specified.
23	**void setColumns(int columns)** Sets the number of columns for this TextArea.
24	**void setFont(Font f)** Sets the current font.
25	**void setLineWrap(boolean wrap)** Sets the line-wrapping policy of the text area.
26	**void setRows(int rows)** Sets the number of rows for this TextArea.
27	**void setTabSize(int size)** Sets the number of characters to expand tabs to.
28	**void setWrapStyleWord(boolean word)** Sets the style of wrapping used, if the text area is wrapping lines.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.text.JTextComponent
- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JTextArea Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;


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

```
import javax.swing.*;

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
      swingControlDemo.showTextAreaDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
```

```
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showTextAreaDemo(){
      headerLabel.setText("Control in action: JTextArea");

      JLabel  commentlabel= new JLabel("Comments: ", JLabel.RIGHT);

      final JTextArea commentTextArea =
         new JTextArea("This is a Swing tutorial "
         +"to make GUI application in Java.",5,20);

      JScrollPane scrollPane = new JScrollPane(commentTextArea);

      JButton showButton = new JButton("Show");

      showButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            statusLabel.setText( commentTextArea.getText());
         }
      });

      controlPanel.add(commentlabel);
      controlPanel.add(scrollPane);
      controlPanel.add(showButton);
      mainFrame.setVisible(true);
   }
}
```

Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.

ImageIcon Class

Introduction

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

Class Declaration

Following is the declaration for **javax.swing.ImageIcon** class –

```
public class ImageIcon
   extends Object
      implements Icon, Serializable, Accessible
```

Field

Following are the fields for **javax.swing.ImageIcon** class –

- **protected static Component component**
- **protected static MediaTracker tracker**

Class Constructors

Sr.No.	Constructor & Description
1	**ImageIcon()** Creates an uninitialized image icon.
2	**ImageIcon(byte[] imageData)** Creates an ImageIcon from an array of bytes which were read from an image file containing a supported image format, such as GIF, JPEG, or (as of 1.3) PNG.
3	**ImageIcon(byte[] imageData, String description)** Creates an ImageIcon from an array of bytes which were read from an image file containing a supported image format, such as GIF, JPEG, or (as of 1.3) PNG.
4	**ImageIcon(Image image)** Creates an ImageIcon from an image object.
5	**ImageIcon(Image image, String description)** Creates an ImageIcon from the image.
6	**ImageIcon(String filename)** Creates an ImageIcon from the specified file.
7	**ImageIcon(String filename, String description)** Creates an ImageIcon from the specified file.
8	**ImageIcon(URL location)** Creates an ImageIcon from the specified URL.

9	**ImageIcon(URL location, String description)** Creates an ImageIcon from the specified URL.

Class Methods

Sr.No.	Method & Description
1	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this ImageIcon.
2	**String getDescription()** Gets the description of the image.
3	**int getIconHeight()** Gets the height of the icon.
4	**int getIconWidth()** Gets the width of the icon.
5	**Image getImage()** Returns this icon's Image.
6	**int getImageLoadStatus()** Returns the status of the image loading operation.
7	**ImageObserver getImageObserver()** Returns the image observer for the image.
8	**protected void loadImage(Image image)** Loads the image, returning only when the image is loaded.
9	**void paintIcon(Component c, Graphics g, int x, int y)** Paints the icon.
10	**void setDescription(String description)**

	Sets the description of the image.
11	**void setImage(Image image)** Sets the image displayed by this icon.
12	**void setImageObserver(ImageObserver observer)** Sets the image observer for the image.
13	**String toString()** Returns a string representation of this image.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

ImageIcon Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }
```

```
   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
      swingControlDemo.showImageIconDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   // Returns an ImageIcon, or null if the path was invalid.
   private static ImageIcon createImageIcon(String path,
      String description) {
      java.net.URL imgURL = SwingControlDemo.class.getResource(path);
      if (imgURL != null) {
         return new ImageIcon(imgURL, description);
      } else {
         System.err.println("Couldn't find file: " + path);
```

```
         return null;
      }
   }
   private void showImageIconDemo(){
      headerLabel.setText("Control in action: ImageIcon");

      ImageIcon icon = createImageIcon("/resources/java_icon.png","Java");

      JLabel commentlabel = new JLabel("", icon,JLabel.CENTER);

      controlPanel.add(commentlabel);

      mainFrame.setVisible(true);
   }
}
```

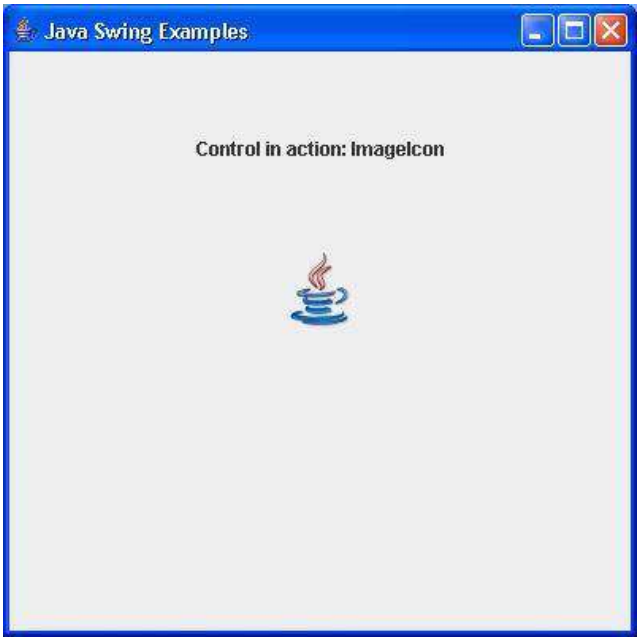
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.



JScrollBar Class

Introduction

The class **JScrollBar** is an implementation of scrollbar.

Class Declaration

Following is the declaration for **javax.swing.JScrollBar** class –

```
public class JScrollBar
   extends JComponent
      implements Adjustable, Accessible
```

Field

Following are the fields for **javax.swing.ImageIcon** class:

- **protected int blockIncrement**
- **protected BoundedRangeModel model** – The model that represents the scrollbar's minimum, maximum, extent (aka "visibleAmount") and current value.
- **protected int orientation**
- **protected int unitIncrement**

Class Constructors

Sr.No.	Constructor & Description
1	**JScrollBar()** Creates a vertical scrollbar with the initial values.
2	**JScrollBar(int orientation)** Creates a scrollbar with the specified orientation and the initial values.
3	**JScrollBar(int orientation, int value, int extent, int min, int max)** Creates a scrollbar with the specified orientation, value, extent, minimum, and maximum.

Class Methods

Sr.No.	Method & Description
1	**void addAdjustmentListener(AdjustmentListener l)** Adds an AdjustmentListener.
2	**protected void fireAdjustmentValueChanged(int id, int type, int value)** Notifies listeners that the scrollbar's model has changed.
3	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JScrollBar.
4	**AdjustmentListener[] getAdjustmentListeners()** Returns an array of all the AdjustmentListeners added to this JScrollBar with addAdjustmentListener().
5	**int getBlockIncrement()** For backwards compatibility with java.awt.Scrollbar.
6	**int getBlockIncrement(int direction)** Returns the amount to change the scrollbar's value by, given a block (usually "page") up/down request.
7	**int getMaximum()** The maximum value of the scrollbar is maximum - extent.
8	**Dimension getMaximumSize()** The scrollbar is flexible along it's scrolling axis and rigid along the other axis.
9	**int getMinimum()** Returns the minimum value supported by the scrollbar (usually zero).
10	**Dimension getMinimumSize()**

	The scrollbar is flexible along it's scrolling axis and rigid along the other axis.

11	**BoundedRangeModel getModel()** Returns data model that handles the scrollbar's four fundamental properties: minimum, maximum, value, extent.
12	**int getOrientation()** Returns the component's orientation (horizontal or vertical).
13	**ScrollBarUI getUI()** Returns the delegate that implements the look and feel for this component.
14	**String getUIClassID()** Returns the name of the LookAndFeel class for this component.
15	**int getUnitIncrement()** For backwards compatibility with java.awt.Scrollbar.
16	**int getUnitIncrement(int direction)** Returns the amount to change the scrollbar's value by, given a unit up/down request.
17	**int getValue()** Returns the scrollbar's value.
18	**boolean getValueIsAdjusting()** True if the scrollbar knob is being dragged.
19	**int getVisibleAmount()** Returns the scrollbar's extent, aka its "visibleAmount".
20	**protected String paramString()** Returns a string representation of this JScrollBar.
21	**void removeAdjustmentListener(AdjustmentListener l)** Removes an AdjustmentEvent listener.

22	**void setBlockIncrement(int blockIncrement)** Sets the blockIncrement property.
23	**void setEnabled(boolean x)** Enables the component so that the knob position can be changed.
24	**void setMaximum(int maximum)** Sets the model's maximum property.
25	**void setMinimum(int minimum)** Sets the model's minimum property.
26	**void setModel(BoundedRangeModel newModel)** Sets the model that handles the scrollbar's four fundamental properties: minimum, maximum, value, extent.
27	**void setOrientation(int orientation)** Set the scrollbar's orientation to either VERTICAL or HORIZONTAL.
28	**void setUI(ScrollBarUI ui)** Sets the L&F object that renders this component.
29	**void setUnitIncrement(int unitIncrement)** Sets the unitIncrement property.
30	**void setValue(int value)** Sets the scrollbar's value.
31	**void setValueIsAdjusting(boolean b)** Sets the model's valueIsAdjusting property.
32	**void setValues(int newValue, int newExtent, int newMin, int newMax)** Sets the four BoundedRangeModel properties after forcing the arguments to obey the usual constraints.

33	**void setVisibleAmount(int extent)** Sets the model's extent property.
34	**void updateUI()** Overrides JComponent.updateUI.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

JScrollBar Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
      swingControlDemo.showScrollbarDemo();
   }
```

```
   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showScrollbarDemo(){
      headerLabel.setText("Control in action: JScrollbar");

      final JScrollBar horizontalScroller =
         new JScrollBar(JScrollBar.HORIZONTAL);
      final JScrollBar verticalScroller = new JScrollBar();
      verticalScroller.setOrientation(JScrollBar.VERTICAL);
      horizontalScroller.setMaximum (100);
      horizontalScroller.setMinimum (1);
      verticalScroller.setMaximum (100);
      verticalScroller.setMinimum (1);

      horizontalScroller.addAdjustmentListener(new AdjustmentListener() {
```

```
         @Override
         public void adjustmentValueChanged(AdjustmentEvent e) {
            statusLabel.setText("Horozontal: "
               +horizontalScroller.getValue()
               +" ,Vertical: "
               + verticalScroller.getValue());
            }
      });

      verticalScroller.addAdjustmentListener(new AdjustmentListener() {

            @Override
            public void adjustmentValueChanged(AdjustmentEvent e) {
               statusLabel.setText("Horozontal: "
               +horizontalScroller.getValue()
               +" ,Vertical: "+ verticalScroller.getValue());
            }
         });

      controlPanel.add(horizontalScroller);
      controlPanel.add(verticalScroller);

      mainFrame.setVisible(true);
   }
}
```

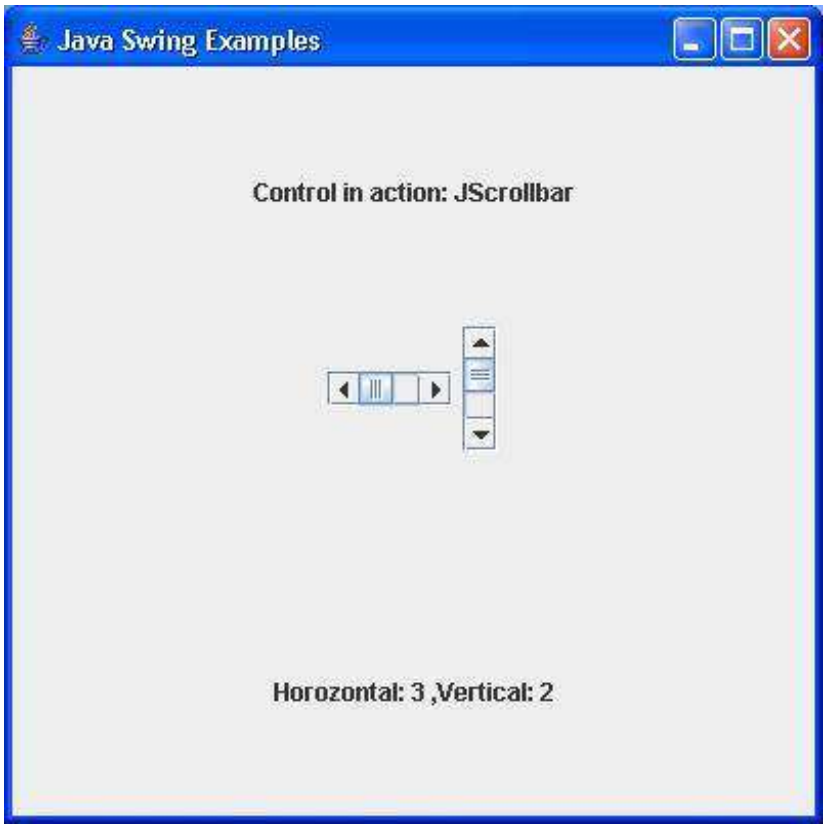
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.



JOptionPane Class

Introduction

The class **JOptionPane** is a component which provides standard methods to pop up a standard dialog box for a value or informs the user of something.

Class Declaration

Following is the declaration for **javax.swing.JOptionPane** class:

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

Field

Following are the fields for **javax.swing.JOptionPane** class:

- **static int CANCEL_OPTION** - Return value from class method if CANCEL is chosen.

- **static int CLOSED_OPTION** - Return value from class method if the user closes window without selecting anything, more than likely this should be treated as either a CANCEL_OPTION or NO_OPTION.
- **static int DEFAULT_OPTION** - Type meaning Look and Feel should not supply any options - only use the options from the JOptionPane.
- **static int ERROR_MESSAGE** - Used for error messages.
- **protected Icon icon** - Icon used in pane.
- **static string ICON_PROPERTY** - Bound property name for icon.
- **static int INFORMATION_MESSAGE** - Used for information messages.
- **static string INITIAL_SELECTION_VALUE_PROPERTY** - Bound property name for initialSelectionValue.
- **static string INITIAL_VALUE_PROPERTY** - Bound property name for initialValue.
- **protected Object initialSelectionValue** - Initial value to select in selectionValues.
- **protected Object initialValue** - Value that should be initially selected in options.
- **static string INPUT_VALUE_PROPERTY** - Bound property name for inputValue.
- **protected Object inputValue** - Value the user has input.
- **protected Object message** - Message to display.
- **static string MESSAGE_PROPERTY** - Bound property name for message.
- **static string MESSAGE_TYPE_PROPERTY** - Bound property name for type.
- **static int OK_CANCEL_OPTION** - Type used for showConfirmDialog.
- **protected int messageType** - Message type.
- **static int NO_OPTION** - Return value from class method, if NO is chosen.
- **static int OK_OPTION** - Return value from class method, if OK is chosen.
- **static string OPTION_TYPE_PROPERTY** - Bound property name for optionType.
- **protected Object[] options** - Options to display to the user.
- **static string OPTIONS_PROPERTY** - Bound property name for option.

- **protected int optionType** - Option type, one of DEFAULT_OPTION, YES_NO_OPTION, YES_NO_CANCEL_OPTION or OK_CANCEL_OPTION.
- **static int PLAIN_MESSAGE** - No icon is used.
- **static int QUESTION_MESSAGE** - Used for questions.
- **static string SELECTION_VALUES_PROPERTY** - Bound property name for selectionValues.
- **protected Object[] selectionValues** - Array of values the user can choose from.
- **static Object UNINITIALIZED_VALUE** - Indicates that the user has not yet selected a value.
- **protected Object value** - Currently selected value, will be a valid option, or UNINITIALIZED_VALUE or null.
- **static string VALUE_PROPERTY** - Bound property name for value.
- **static string WANTS_INPUT_PROPERTY** - Bound property name for wantsInput.
- **protected boolean wantsInput** - If true, a UI widget will be provided to the user to get input.
- **static int WARNING_MESSAGE** - Used for warning messages.
- **static int YES_NO_CANCEL_OPTION** - Type used for showConfirmDialog.
- **static int YES_NO_OPTION** - Type used for showConfirmDialog.
- **static int YES_OPTION** - Return value from class method, if YES is chosen.

Class Constructors

Sr.No.	Constructor & Description
1	**JOptionPane()** Creates a JOptionPane with a test message.
2	**JOptionPane(Object message)** Creates a instance of JOptionPane to display a message using the plain-message message type and the default options delivered by the UI.
3	**JOptionPane(Object message, int messageType)**

	Creates an instance of JOptionPane to display a message with the specified message type and the default options.
4	**JOptionPane(Object message, int messageType, int optionType)** Creates an instance of JOptionPane to display a message with the specified message type and options.
5	**JOptionPane(Object message, int messageType, int optionType, Icon icon)** Creates an instance of JOptionPane to display a message with the specified message type, options, and icon.
6	**JOptionPane(Object message, int messageType, int optionType, Icon icon, Object[] options)** Creates an instance of JOptionPane to display a message with the specified message type, icon, and options.
7	**JOptionPane(Object message, int messageType, int optionType, Icon icon, Object[] options, Object initialValue)** Creates an instance of JOptionPane to display a message with the specified message type, icon, and options, with the initially-selected option specified.

Class Methods

Sr.No.	Method & Description
1	**JDialog createDialog(Component parentComponent, String title)** Creates and returns a new JDialog wrapping, centered on the parentComponent in the parentComponent's frame.
2	**JDialog createDialog(String title)** Creates and returns a new parentless JDialog with the specified title.
3	**JInternalFrame createInternalFrame(Component parentComponent, String title)** Creates and returns an instance of JInternalFrame.

4	**AccessibleContext getAccessibleContext()** Returns the AccessibleContext associated with this JOptionPane.
5	**static JDesktopPane getDesktopPaneForComponent(Component parentComponent)** Returns the specified component's desktop pane.
6	**static Frame getFrameForComponent(Component parentComponent)** Returns the specified component's Frame.
7	**Icon getIcon()** Returns the icon this pane displays.
8	**Object getInitialSelectionValue()** Returns the input value that is displayed as initially selected by the user.
9	**Object getInitialValue()** Returns the initial value.
10	**Object getInputValue()** Returns the value the user has input, if wantsInput is true.
11	**int getMaxCharactersPerLineCount()** Returns the maximum number of characters to place on a line in a message.
12	**Object getMessage()** Returns the message-object this pane displays.
13	**int getMessageType()** Returns the message type.
14	**Object[] getOptions()** Returns the choices the user can make.
15	**int getOptionType()** Returns the type of options that are displayed.
16	**static Frame getRootFrame()**

	Returns the Frame to use for the class methods in which a frame is not provided.
17	**Object[] getSelectionValues()** Returns the input selection values.
18	**OptionPaneUI getUI()** Returns the UI object which implements the L&F for this component.
19	**String getUIClassID()** Returns the name of the UI class that implements the L&F for this component.
20	**Object getValue()** Returns the value the user has selected.
21	**boolean getWantsInput()** Returns the value of the wantsInput property.
22	**protected String paramString()** Returns a string representation of this JOptionPane.
23	**void selectInitialValue()** Requests that the initial value be selected, which will set focus to the initial value.
24	**void setIcon(Icon newIcon)** Sets the icon to display.
25	**void setInitialSelectionValue(Object newValue)** Sets the input value that is initially displayed as selected by the user.
26	**void setInitialValue(Object newInitialValue)** Sets the initial value that is to be enabled - the Component that has the focus when the pane is initially displayed.
27	**void setInputValue(Object newValue)** Sets the input value that was selected or input by the user.
28	**void setMessage(Object newMessage)**

	Sets the option pane's message-object.
29	**void setMessageType(int newType)** Sets the option pane's message type.
30	**void setOptions(Object[] newOptions)** Sets the options this pane displays.
31	**void setOptionType(int newType)** Sets the options to display.
32	**static voidsetRootFrame(Frame newRootFrame)** Sets the frame to use for class methods in which a frame is not provided.
33	**void setSelectionValues(Object[] newValues)** Sets the input selection values for a pane that provides the user with a list of items to choose from.
34	**void setUI(OptionPaneUI ui)** Sets the UI object which implements the L&F for this component.
35	**void setValue(Object newValue)** Sets the value the user has chosen.
36	**void setWantsInput(boolean newValue)** Sets the wantsInput property.
37	**static int showConfirmDialog(Component parentComponent, Object message)** Brings up a dialog with the options Yes, No and Cancel; with the title, Select an Option.
38	**static int showConfirmDialog(Component parentComponent, Object message, String title, int optionType)** Brings up a dialog where the number of choices is determined by the optionType parameter.
39	

	static int showConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType) Brings up a dialog where the number of choices is determined by the optionType parameter, where the messageType parameter determines the icon to display.
40	**static int showConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon)** Brings up a dialog with a specified icon, where the number of choices is determined by the optionType parameter.
41	**static String showInputDialog(Component parentComponent, Object message)** Shows a question-message dialog requesting input from the user parented to parentComponent.
42	**static String showInputDialog(Component parentComponent, Object message, Object initialSelectionValue)** Shows a question-message dialog requesting input from the user and parented to parentComponent.
43	**static String showInputDialog(Component parentComponent, Object message, String title, int essageType)** Shows a dialog requesting input from the user parented to parentComponent with the dialog having the title title and message type **messageType**.
44	**static Object showInputDialog(Component parentComponent, Object message, String title, int messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue)** Prompts the user for input in a blocking dialog where the initial selection, possible selections, and all other options can be specified.
45	**static String showInputDialog(Object message)** Shows a question-message dialog requesting input from the user.
46	**static String showInputDialog(Object message, Object initialSelectionValue)** Shows a question-message dialog requesting input from the user, with the input value initialized to initialSelectionValue.
47	**static int showInternalConfirmDialog(Component parentComponent, Object message)**

	Brings up an internal dialog panel with the options Yes, No and Cancel; with the title, Select an Option.
48	**static int showInternalConfirmDialog(Component parentComponent, Object message, String title, int optionType)** Brings up an internal dialog panel where the number of choices is determined by the optionType parameter.
49	**static int showInternalConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType)** Brings up an internal dialog panel where the number of choices is determined by the optionType parameter, where the messageType parameter determines the icon to display.
50	**static int showInternalConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon)** Brings up an internal dialog panel with a specified icon, where the number of choices is determined by the optionType parameter.
51	**static String showInternalInputDialog(Component parentComponent, Object message)** Shows an internal question-message dialog requesting input from the user parented to parentComponent.
52	**static String showInternalInputDialog(Component parentComponent, Object message, String title, int messageType)** Shows an internal dialog requesting input from the user parented to parentComponent with the dialog having the title **title** and message type **messageType**.
53	**static Object showInternalInputDialog(Component parentComponent, Object message, String title, int messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue)** Prompts the user for input in a blocking internal dialog where the initial selection, possible selections, and all other options can be specified.
54	**static voidshowInternalMessageDialog(Component parentComponent, Object message)** Brings up an internal confirmation dialog panel.
55	

	static voidshowInternalMessageDialog(Component parentComponent, Object message, String title, int messageType) Brings up an internal dialog panel that displays a message using a default icon determined by the messageType parameter.
56	**static voidshowInternalMessageDialog(Component parentComponent, Object message, String title, int messageType, Icon icon)** Brings up an internal dialog panel displaying a message, specifying all parameters.
57	**static voidshowMessageDialog(Component parentComponent, Object message)** Brings up an information-message dialog titled "Message".
58	**static voidshowMessageDialog(Component parentComponent, Object message, String title, int messageType)** Brings up a dialog that displays a message using a default icon determined by the messageType parameter.
59	**static voidshowMessageDialog(Component parentComponent, Object message, String title, int messageType, Icon icon)** Brings up a dialog displaying a message, specifying all parameters.
60	**static int showOptionDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue)** Brings up a dialog with a specified icon, where the initial choice is determined by the initialValue parameter and the number of choices is determined by the optionType parameter.
61	**void updateUI()** Notification from the UIManager that the L&F has changed.
62	**static int showInternalOptionDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue)** Brings up an internal dialog panel with a specified icon, where the initial choice is determined by the initialValue parameter and the number of choices is determined by the optionType parameter.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JOptionPane Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
      swingControlDemo.showDialogDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
```

```
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showDialogDemo(){
      headerLabel.setText("Control in action: JOptionPane");

      JButton okButton = new JButton("OK");
      JButton javaButton = new JButton("Yes/No");
      JButton cancelButton = new JButton("Yes/No/Cancel");

      okButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(
            mainFrame, "Welcome to TutorialsPoint.com");
         }
      });

      javaButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            int output = JOptionPane.showConfirmDialog(mainFrame
               , "Click any button"
               ,"TutorialsPoint.com"
               ,JOptionPane.YES_NO_OPTION);
```

```
            if(output == JOptionPane.YES_OPTION){
               statusLabel.setText("Yes selected.");
            }else if(output == JOptionPane.NO_OPTION){
               statusLabel.setText("No selected.");
            }
         }
      });

      cancelButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            int output = JOptionPane.showConfirmDialog(mainFrame
               , "Click any button"
               ,"TutorialsPoint.com"
               ,JOptionPane.YES_NO_CANCEL_OPTION,
               JOptionPane.INFORMATION_MESSAGE);

            if(output == JOptionPane.YES_OPTION){
               statusLabel.setText("Yes selected.");
            }else if(output == JOptionPane.NO_OPTION){
               statusLabel.setText("No selected.");
            }else if(output == JOptionPane.CANCEL_OPTION){
               statusLabel.setText("Cancel selected.");
            }
         }
      });


      controlPanel.add(okButton);
      controlPanel.add(javaButton);
      controlPanel.add(cancelButton);
      mainFrame.setVisible(true);
   }
}
```

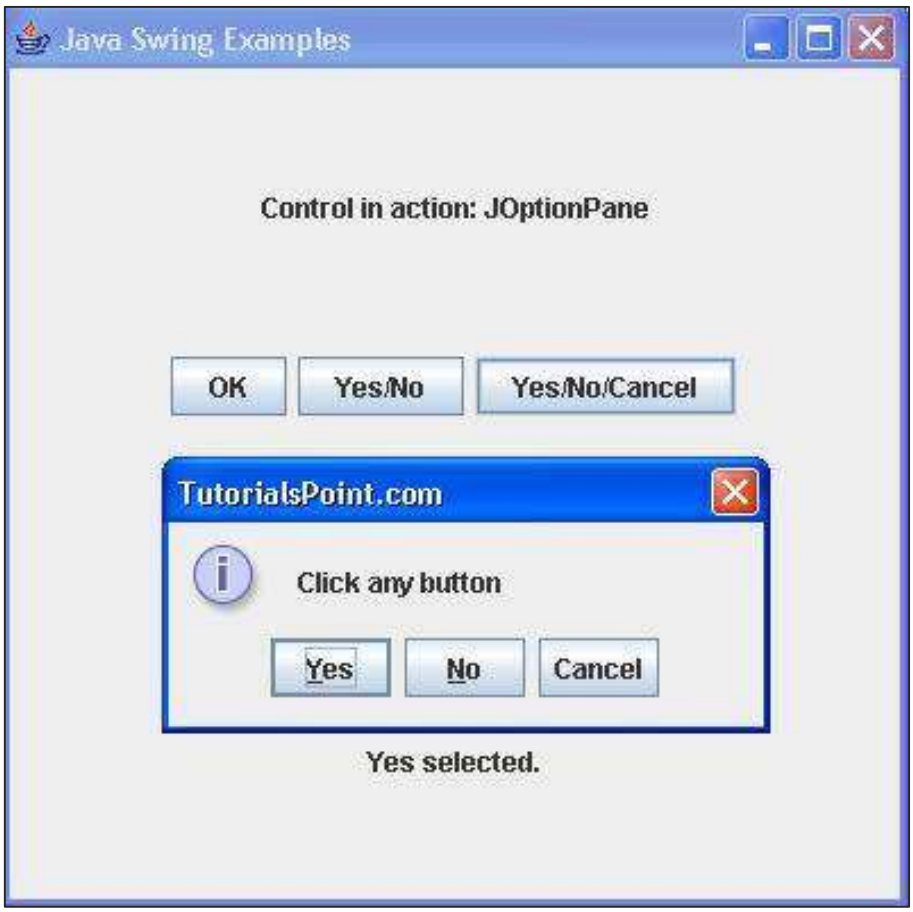
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.



JFileChooser Class

Introduction

The class **JFileChooser** is a component, which provides a simple mechanism for the user to choose a file.

Class Declaration

Following is the declaration for **javax.swing.JFileChooser** class:

```
public class JFileChooser
   extends JComponent
```

```
    implements Accessible
```

Field

Following are the fields for **javax.swing.JFileChooser** class:

- **static String ACCEPT_ALL_FILE_FILTER_USED_CHANGED_PROPERTY** - Identifies whether the AcceptAllFileFilter is used or not.
- **protected AccessibleContext accessibleContext**
- **static String ACCESSORY_CHANGED_PROPERTY** - Says that a different accessory component is in use (for example, to preview files).
- **static String APPROVE_BUTTON_MNEMONIC_CHANGED_PROPERTY** - Identifies change in the mnemonic for the approve (yes, ok) button.
- **static String APPROVE_BUTTON_TEXT_CHANGED_PROPERTY** - Identifies change in the text on the approve (yes, ok) button.
- **static String APPROVE_BUTTON_TOOL_TIP_TEXT_CHANGED_PROPERTY** - Identifies change in the tooltip text for the approve (yes, ok) button.
- **static int APPROVE_OPTION** - Return value if approve (yes, ok) is chosen.
- **static String APPROVE_SELECTION** - Instruction to approve the current selection (same as pressing yes or ok).
- **static int CANCEL_OPTION** - Return value if cancel is chosen.
- **static String CANCEL_SELECTION** - Instruction to cancel the current selection.
- **static String CHOOSABLE_FILE_FILTER_CHANGED_PROPERTY** - Identifies a change in the list of predefined file filters the user can choose from.
- **static String CONTROL_BUTTONS_ARE_SHOWN_CHANGED_PROPERTY** - Instruction to display the control buttons.
- **static int CUSTOM_DIALOG** - Type value indicating the JFileChooser supports a developer-specified file operation.
- **static String DIALOG_TITLE_CHANGED_PROPERTY** - Identifies a change in the dialog title.
- **static String DIALOG_TYPE_CHANGED_PROPERTY** - Identifies a change in the type of files displayed (files only, directories only, or both files and directories).
- **static int DIRECTORIES_ONLY** - Instruction to display only directories.

- **static String DIRECTORY_CHANGED_PROPERTY** - Identifies the user's directory change.
- **static int ERROR_OPTION** - Returns value if an error occured.
- **static String FILE_FILTER_CHANGED_PROPERTY** – Identifies the user changed the kind of files to display.
- **static String FILE_HIDING_CHANGED_PROPERTY** - Identifies a change in the display-hidden-files property.
- **static String FILE_SELECTION_MODE_CHANGED_PROPERTY** - Identifies a change in the kind of selection (single, multiple, etc.)
- **static String FILE_SYSTEM_VIEW_CHANGED_PROPERTY** - Says that a different object is being used to find available drives on the system.
- **static String FILE_VIEW_CHANGED_PROPERTY** - Says that a different object is being used to retrieve file information.
- **static int FILES_AND_DIRECTORIES** - Instruction to display both files and directories.
- **static int FILES_ONLY** - Instruction to display only files.
- **static String MULTI_SELECTION_ENABLED_CHANGED_PROPERTY** - Enables multiple-file selections.
- **static int OPEN_DIALOG** - Type value indicating that the JFileChooser supports an "Open" file operation.
- **static int SAVE_DIALOG** - Type value indicating that the JFileChooser supports a "Save" file operation.
- **static String SELECTED_FILE_CHANGED_PROPERTY** - Identifies change in the user's single-file selection.
- **static String SELECTED_FILES_CHANGED_PROPERTY** - Identifies change in the user's multiple-file selection.

Class Constructors

Sr.No.	Constructor & Description
1	**JFileChooser()** Constructs a JFileChooser pointing to the user's default directory.
2	**JFileChooser(File currentDirectory)**

	Constructs a JFileChooser using the given File as the path.
3	**JFileChooser(File currentDirectory, FileSystemView fsv)** Constructs a JFileChooser using the given current directory and FileSystemView.
4	**JFileChooser(FileSystemView fsv)** Constructs a JFileChooser using the given FileSystemView.
5	**JFileChooser(String currentDirectoryPath)** Constructs a JFileChooser using the given path.
6	**JFileChooser(String currentDirectoryPath, FileSystemView fsv)** Constructs a JFileChooser using the given current directory path and FileSystemView.

Class Methods

Sr.No.	Method & Description
1	**boolean accept(File f)** Returns true if the file should be displayed.
2	**void addActionListener(ActionListener l)** Adds an ActionListener to the file chooser.
3	**void addChoosableFileFilter(FileFilter filter)** Adds a filter to the list of user choosable file filters.
4	**void approveSelection()** Called by the UI when the user hits the Approve button (labeled "Open" or "Save", by default).
5	**void cancelSelection()** Called by the UI when the user chooses the Cancel button.
6	

	void changeToParentDirectory() Changes the directory to be set to the parent of the current directory.
7	**protected JDialog createDialog(Component parent)** Creates and returns a new JDialog wrapping, centered on the parent in the parent's frame.
8	**void ensureFileIsVisible(File f)** Makes sure that the specified file is viewable, and not hidden.
9	**protected void fireActionPerformed(String command)** Notifies all listeners that have registered interest for notification on this event type.
10	**FileFilter getAcceptAllFileFilter()** Returns the AcceptAll file filter.
11	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JFileChooser.
12	**JComponent getAccessory()** Returns the accessory component.
13	**ActionListener[]getActionListeners()** Returns an array of all the action listeners registered on this file chooser.
14	**int getApproveButtonMnemonic()** Returns the approve button's mnemonic.
15	**String getApproveButtonText()** Returns the text used in the ApproveButton in the FileChooserUI.
16	**String getApproveButtonToolTipText()** Returns the tooltip text used in the ApproveButton.
17	**FileFilter[] getChoosableFileFilters()** Gets the list of user choosable file filters.

18	**boolean getControlButtonsAreShown()** Returns the value of the controlButtonsAreShown property.
19	**File getCurrentDirectory()** Returns the current directory.
20	**String getDescription(File f)** Returns the file description.
21	**String getDialogTitle()** Gets the string that goes in the JFileChooser's titlebar.
22	**int getDialogType()** Returns the type of this dialog.
23	**boolean getDragEnabled()** Gets the value of the dragEnabled property.
24	**FileFilter getFileFilter()** Returns the currently selected file filter.
25	**int getFileSelectionMode()** Returns the current file-selection mode.
26	**FileSystemView getFileSystemView()** Returns the file system view.
27	**FileView getFileView()** Returns the current file view.
28	**Icon getIcon(File f)** Returns the icon for this file or type of file, depending on the system.
29	**String getName(File f)** Returns the filename.
30	**File getSelectedFile()**

	Returns the selected file.
31	**File[] getSelectedFiles()** Returns a list of selected files, if the file chooser is set to allow multiple selection.
32	**String getTypeDescription(File f)** Returns the file type.
33	**FileChooserUI getUI()** Gets the UI object which implements the L&F for this component.
34	**String getUIClassID()** Returns a string that specifies the name of the L&F class which renders this component.
35	**boolean isAcceptAllFileFilterUsed()** Returns whether the AcceptAll FileFilter is used.
36	**boolean isDirectorySelectionEnabled()** Convenience call that determines if the directories are selectable based on the current file selection mode.
37	**boolean isFileHidingEnabled()** Returns true if the hidden files are not shown in the file chooser; otherwise, returns false.
38	**boolean isFileSelectionEnabled()** Convenience call that determines if the files are selectable based on the current file selection mode.
39	**boolean isMultiSelectionEnabled()** Returns true if multiple files can be selected.
40	**boolean isTraversable(File f)** Returns true if the file (directory) can be visited.
41	**protected String paramString()** Returns a string representation of this JFileChooser.

42	**void removeActionListener(ActionListener l)** Removes an ActionListener from the file chooser.
43	**boolean removeChoosableFileFilter(FileFilter f)** Removes a filter from the list of user choosable file filters.
44	**void rescanCurrentDirectory()** Tells the UI to rescan its files list from the current directory.
45	**void resetChoosableFileFilters()** Resets the choosable file filter list to its starting state.
46	**void setAcceptAllFileFilterUsed(boolean b)** Determines whether the AcceptAll FileFilter is used as an available choice in the choosable filter list.
47	**void setAccessory(JComponent newAccessory)** Sets the accessory component.
48	**void setApproveButtonMnemonic(char mnemonic)** Sets the approve button's mnemonic using a character.
49	**void setApproveButtonMnemonic(int mnemonic)** Sets the approve button's mnemonic using a numeric keycode.
50	**void setApproveButtonText(String approveButtonText)** Sets the text used in the ApproveButton in the FileChooserUI.
51	**void setApproveButtonToolTipText(String toolTipText)** Sets the tooltip text used in the ApproveButton.
52	**void setControlButtonsAreShown(boolean b)** Sets the property that indicates whether the approve and cancel buttons are shown in the file chooser.
53	**void setCurrentDirectory(File dir)**

	Sets the current directory.
54	**void setDialogTitle(String dialogTitle)** Sets the string that goes in the JFileChooser window's title bar.
55	**void setDialogType(int dialogType)** Sets the type of this dialog.
56	**void setDragEnabled(boolean b)** Sets the dragEnabled property, which must be true to enable automatic drag handling (the first part of drag and drop) on this component.
57	**void setFileFilter(FileFilter filter)** Sets the current file filter.
58	**void setFileHidingEnabled(boolean b)** Sets file hiding on or off.
59	**void setFileSelectionMode(int mode)** Sets the JFileChooser to allow the user to just select files, just select directories, or select both files and directories.
60	**void setFileSystemView(FileSystemView fsv)** Sets the file system view that the JFileChooser uses for accessing and creating file system resources, such as finding the floppy drive and getting a list of root drives.
61	**void setFileView(FileView fileView)** Sets the file view to be used to retrieve UI information, such as the icon that represents a file or the type description of a file.
62	**void setMultiSelectionEnabled(boolean b)** Sets the file chooser to allow multiple file selections.
63	**void setSelectedFile(File file)** Sets the selected file.
64	**void setSelectedFiles(File[] selectedFiles)**

	Sets the list of selected files if the file chooser is set to allow multiple selection.
65	**protected void setup(FileSystemView view)** Performs common constructor initialization and setup.
66	**int showDialog(Component parent, String approveButtonText)** Pops a custom file chooser dialog with a custom approve button.
67	**int showOpenDialog(Component parent)** Pops up an "Open File" file chooser dialog.
68	**int showSaveDialog(Component parent)** Pops up a "Save File" file chooser dialog.
69	**void updateUI()** Resets the UI property to a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JFileChooser Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingControlDemo {
```

```
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
      swingControlDemo.showFileChooserDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }
```

```
   private void showFileChooserDemo(){
      headerLabel.setText("Control in action: JFileChooser");

      final JFileChooser  fileDialog = new JFileChooser();
      JButton showFileDialogButton = new JButton("Open File");
      showFileDialogButton.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            int returnVal = fileDialog.showOpenDialog(mainFrame);
            if (returnVal == JFileChooser.APPROVE_OPTION) {
               java.io.File file = fileDialog.getSelectedFile();
               statusLabel.setText("File Selected :"
               + file.getName());
            }
            else{
               statusLabel.setText("Open command cancelled by user." );
            }
         }
      });
      controlPanel.add(showFileDialogButton);
      mainFrame.setVisible(true);
   }
}
```

Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.

JProgressBar Class

Introduction

The class **JProgressBar** is a component which visually displays the progress of some task.

Class Declaration

Following is the declaration for **javax.swing.JProgressBar** class:

```
public class JProgressBar
   extends JComponent
      implements SwingConstants, Accessible
```

Field

Following are the fields for **javax.swing.JProgressBar** class:

- **protected ChangeEvent changeEvent** - Only one ChangeEvent is needed per instance since the event's only interesting property is the immutable source, which is the progress bar.
- **protected ChangeListener changeListener** - Listens for change events sent by the progress bar's model, redispatching them to change-event listeners registered upon this progress bar.
- **protected BoundedRangeModel model** - The object that holds the data for the progress bar.
- **protected int orientation** - Whether the progress bar is horizontal or vertical.
- **protected boolean paintBorder** - Whether to display a border around the progress bar.
- **protected boolean paintString** - Whether to display a string of text on the progress bar.
- **protected String progressString** - An optional string that can be displayed on the progress bar.

Class Constructors

Sr.No.	Constructor & Description
1	**JProgressBar()** Creates a horizontal progress bar that displays a border but no progress string.
2	**JProgressBar(BoundedRangeModel newModel)** Creates a horizontal progress bar that uses the specified model to hold the progress bar's data.
3	**JProgressBar(int orient)** Creates a progress bar with the specified orientation, which can be either SwingConstants. VERTICAL or SwingConstants.HORIZONTAL.
4	**JProgressBar(int min, int max)** Creates a horizontal progress bar with the specified minimum and maximum.
5	**JProgressBar(int orient, int min, int max)** Creates a progress bar using the specified orientation, minimum, and maximum.

Class Methods

Sr.No.	Method & Description
1	**void addChangeListener(ChangeListener l)** Adds the specified ChangeListener to the progress bar.
2	**protected ChangeListener createChangeListener()** Subclasses that want to handle change events from the model differently can override this to return an instance of a custom ChangeListener implementation.
3	**protected void fireStateChanged()** Send a ChangeEvent, whose source is JProgressBar, to all ChangeListeners that have registered interest in ChangeEvents.
4	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JProgressBar.
5	**ChangeListener[] getChangeListeners()** Returns an array of all the ChangeListeners added to this progress bar with addChangeListener.
6	**int getMaximum()** Returns the progress bar's maximum value from the BoundedRangeModel.
7	**int getMinimum()** Returns the progress bar's minimum value from the BoundedRangeModel.
8	**BoundedRangeModel getModel()** Returns the data model used by this progress bar.
9	**int getOrientation()**

	Returns SwingConstants.VERTICAL or SwingConstants.HORIZONTAL, depending on the orientation of the progress bar.

10	**double getPercentComplete()** Returns the percent complete for the progress bar.
11	**String getString()** Returns a String representation of the current progress.
12	**ProgressBarUI getUI()** Returns the look-and-feel object that renders this component.
13	**String getUIClassID()** Returns the name of the look-and-feel class that renders this component.
14	**int getValue()** Returns the progress bar's current value from the BoundedRangeModel.
15	**boolean isBorderPainted()** Returns the borderPainted property.
16	**boolean isIndeterminate()** Returns the value of the indeterminate property.
17	**boolean isStringPainted()** Returns the value of the stringPainted property.
18	**protected void paintBorder(Graphics g)** Paints the progress bar's border, if the borderPainted property is true.
19	**protected String paramString()** Returns a string representation of this JProgressBar.
20	**void removeChangeListener(ChangeListener l)** Removes a ChangeListener from the progress bar.
21	**void setBorderPainted(boolean b)** Sets the borderPainted property, which is true if the progress bar should paint its border.
22	**void setIndeterminate(boolean newValue)**

	Sets the indeterminate property of the progress bar, which determines whether the progress bar is in determinate or indeterminate mode.
23	**void setMaximum(int n)** Sets the progress bar's maximum value (stored in the progress bar's data model) to **n**.
24	**void setMinimum(int n)** Sets the progress bar's minimum value (stored in the progress bar's data model) to **n**.
25	**void setModel(BoundedRangeModel newModel)** Sets the data model used by the JProgressBar.
26	**void setOrientation(int newOrientation)** Sets the progress bar's orientation to newOrientation, which must be SwingConstants.VERTICAL or SwingConstants.HORIZONTAL.
27	**void setString(String s)** Sets the value of the progress string.
28	**void setStringPainted(boolean b)** Sets the value of the stringPainted property, which determines whether the progress bar should render a progress string.
29	**void setUI(ProgressBarUI ui)** Sets the look-and-feel object that renders this component.
30	**void setValue(int n)** Sets the progress bar's current value to **n**.
31	**void updateUI()** Resets the UI property to a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.JComponent

- java.awt.Container
- java.awt.Component
- java.lang.Object

JProgressBar Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
      swingControlDemo.showProgressBarDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
```

```
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private JProgressBar progressBar;
   private Task task;
   private JButton startButton;
   private JTextArea outputTextArea;

   private void showProgressBarDemo(){
      headerLabel.setText("Control in action: JProgressBar");

      progressBar = new JProgressBar(0, 100);
      progressBar.setValue(0);
      progressBar.setStringPainted(true);
      startButton = new JButton("Start");

      outputTextArea = new JTextArea("",5,20);

      JScrollPane scrollPane = new JScrollPane(outputTextArea);
         startButton.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            task = new Task();
            task.start();
```

```
         }});

      controlPanel.add(startButton);
      controlPanel.add(progressBar);
      controlPanel.add(scrollPane);
      mainFrame.setVisible(true);
   }

   private class Task extends Thread {
      public Task(){
      }

      public void run(){
         for(int i =0; i<= 100; i+=10){
            final int progress = i;
            SwingUtilities.invokeLater(new Runnable() {
               public void run() {
                  progressBar.setValue(progress);
                  outputTextArea.setText(outputTextArea.getText()
                  + String.format("Completed %d%% of task.\n", progress));
               }
            });
            try {
               Thread.sleep(100);
            } catch (InterruptedException e) {}
         }
      }
   }
}
```

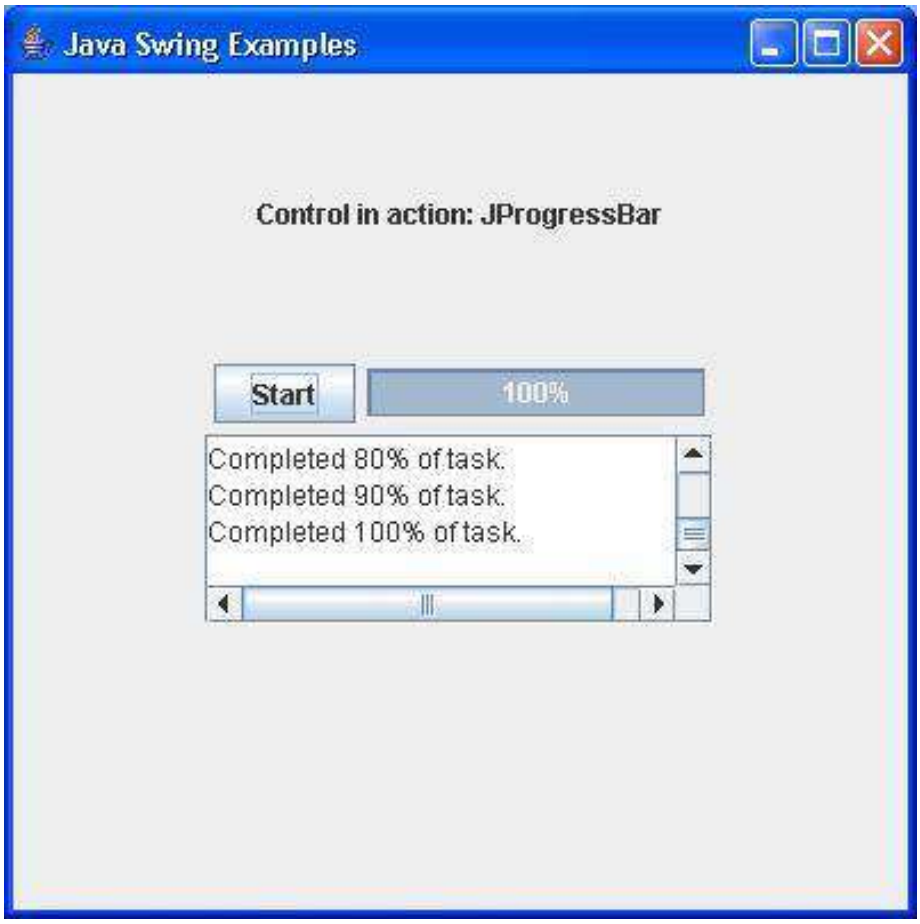
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.



JSlider Class

Introduction

The class **JSlider** is a component which lets the user graphically select a value by sliding a knob within a bounded interval.

Class Declaration

Following is the declaration for **javax.swing.JSlider** class:

```
public class JSlider
   extends JComponent
      implements SwingConstants, Accessible
```

Field

Following are the fields for **javax.swing.JSlider** class:

- **protected ChangeEvent changeEvent** - Only one ChangeEvent is needed per slider instance since the event's only (read-only) state is the source property.
- **protected ChangeListener changeListener** - The changeListener (no suffix) is the listener we add to the slider's model.
- **protected int majorTickSpacing** - The number of values between the major tick marks - the larger marks that break up the minor tick marks.
- **protected int minorTickSpacing** - The number of values between the minor tick marks - the smaller marks that occur between the major tick marks.
- **protected int orientation** - Whether the slider is horizontal or vertical. The default is horizontal.
- **protected BoundedRangeModel sliderModel** - The data model that handles the numeric maximum value, minimum value, and current-position value for the slider.
- **protected boolean snapToTicks** - If true, the knob (and the data value it represents) resolves to the closest tick mark next to where the user positioned the knob.

Class Constructors

Sr.No.	Constructor & Description
1	**JSlider()** Creates a horizontal slider with the range 0 to 100 and an initial value of 50.
2	**JSlider(BoundedRangeModel brm)** Creates a horizontal slider using the specified BoundedRangeModel.
3	**JSlider(int orientation)** Creates a slider using the specified orientation with the range 0 to 100 and an initial value of 50.
4	**JSlider(int min, int max)** Creates a horizontal slider using the specified min and max with an initial value equal to the average of the min plus max.
5	**JSlider(int min, int max, int value)**

	Creates a horizontal slider using the specified min, max and value.
6	**JSlider(int orientation, int min, int max, int value)** Creates a slider with the specified orientation and the specified minimum, maximum, and initial values.

Class Methods

Sr.No.	Method & Description
1	**void addChangeListener(ChangeListener l)** Adds a ChangeListener to the slider.
2	**protected ChangeListener createChangeListener()** Subclasses that want to handle ChangeEvents from the model differently can override this to return an instance of a custom ChangeListener implementation.
3	**Hashtable createStandardLabels(int increment)** Creates a Hashtable of numerical text labels, starting at the slider minimum, and using the increment specified.
4	**Hashtable createStandardLabels(int increment, int start)** Creates a Hashtable of numerical text labels, starting at the starting point specified, and using the increment specified.
5	**protected void fireStateChanged()** Send a ChangeEvent, whose source is JSlider, to all ChangeListeners that have registered interest in ChangeEvents.
6	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with JSlider.
7	**ChangeListener[] getChangeListeners()** Returns an array of all the ChangeListeners added to JSlider with addChangeListener().
8	**int getExtent()**

	Returns the "extent" from the BoundedRangeModel.
9	**boolean getInverted()** Returns true if the value-range shown for the slider is reversed.
10	**Dictionary getLabelTable()** Returns the dictionary of what labels to draw at which values.
11	**int getMajorTickSpacing()** This method returns the major tick spacing.
12	**int getMaximum()** Returns the maximum value supported by the slider from the BoundedRangeModel.
13	**int getMinimum()** Returns the minimum value supported by the slider from the BoundedRangeModel.
14	**int getMinorTickSpacing()** This method returns the minor tick spacing.
15	**BoundedRangeModel getModel()** Returns the BoundedRangeModel that handles the slider's three fundamental properties: minimum, maximum, value.
16	**int getOrientation()** Return this slider's vertical or horizontal orientation.
17	**boolean getPaintLabels()** Tells if labels are to be painted.
18	**boolean getPaintTicks()** Tells if tick marks are to be painted.
19	**boolean getPaintTrack()** Tells if the track (area the slider slides in) is to be painted.
20	**boolean getSnapToTicks()**

	Returns true if the knob (and the data value it represents) resolves to the closest tick mark next to where the user positioned the knob.
21	**SliderUI getUI()** Gets the UI object which implements the L&F for this component.
22	**String getUIClassID()** Returns the name of the L&F class that renders this component.
23	**int getValue()** Returns the slider's current value from the BoundedRangeModel.
24	**boolean getValueIsAdjusting()** Returns the valueIsAdjusting property from the model.
25	**protected String paramString()** Returns a string representation of this JSlider.
26	**void removeChangeListener(ChangeListener l)** Removes a ChangeListener from the slider.
27	**void setExtent(int extent)** Sets the size of the range "covered" by the knob.
28	**void setFont(Font font)** Sets the font for this component.
29	**void setInverted(boolean b)** Specifies true to reverse the value-range shown for the slider and false to put the value range in the normal order.
30	**void setLabelTable(Dictionary labels)** Used to specify what label will be drawn at any given value.
31	**void setMajorTickSpacing(int n)** This method sets the major tick spacing.
32	**void setMaximum(int maximum)**

	Sets the slider's maximum value to maximum.
33	**void setMinimum(int minimum)** Sets the slider's minimum value to minimum.
34	**void setMinorTickSpacing(int n)** This method sets the minor tick spacing.
35	**void setModel(BoundedRangeModel newModel)** Sets the BoundedRangeModel that handles the slider's three fundamental properties: minimum, maximum, value.
36	**void setOrientation(int orientation)** Set the slider's orientation to either SwingConstants.VERTICAL or SwingConstants.HORIZONTAL.
37	**void setPaintLabels(boolean b)** Determines whether labels are painted on the slider.
38	**void setPaintTicks(boolean b)** Determines whether tick marks are painted on the slider.
39	**void setPaintTrack(boolean b)** Determines whether the track is painted on the slider.
40	**void setSnapToTicks(boolean b)** Specifying true makes the knob (and the data value it represents) resolve to the closest tick mark next to where the user positioned the knob.
41	**void setUI(SliderUI ui)** Sets the UI object which implements the L&F for this component.
42	**void setValue(int n) Sets the slider's current value to n**
43	**void setValueIsAdjusting(boolean b)** Sets the model's valueIsAdjusting property.
44	**protected void updateLabelUIs()**

	Updates the UIs for the labels in the label table by calling updateUI on each label.
45	**void updateUI()** Resets the UI property to a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JSlider Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class SwingControlDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
      swingControlDemo.showSliderDemo();
```

```
}

private void prepareGUI(){
   mainFrame = new JFrame("Java Swing Examples");
   mainFrame.setSize(400,400);
   mainFrame.setLayout(new GridLayout(3, 1));
   mainFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent windowEvent){
         System.exit(0);
      }
   });
   headerLabel = new JLabel("", JLabel.CENTER);
   statusLabel = new JLabel("",JLabel.CENTER);

   statusLabel.setSize(350,100);

   controlPanel = new JPanel();
   controlPanel.setLayout(new FlowLayout());

   mainFrame.add(headerLabel);
   mainFrame.add(controlPanel);
   mainFrame.add(statusLabel);
   mainFrame.setVisible(true);
}

private void showSliderDemo(){
   headerLabel.setText("Control in action: JSlider");
   JSlider slider= new JSlider(JSlider.HORIZONTAL,0,100,10);
   slider.addChangeListener(new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
         statusLabel.setText("Value : "
         + ((JSlider)e.getSource()).getValue());
      }
   });
   controlPanel.add(slider);
   mainFrame.setVisible(true);
}
```

```
}
```

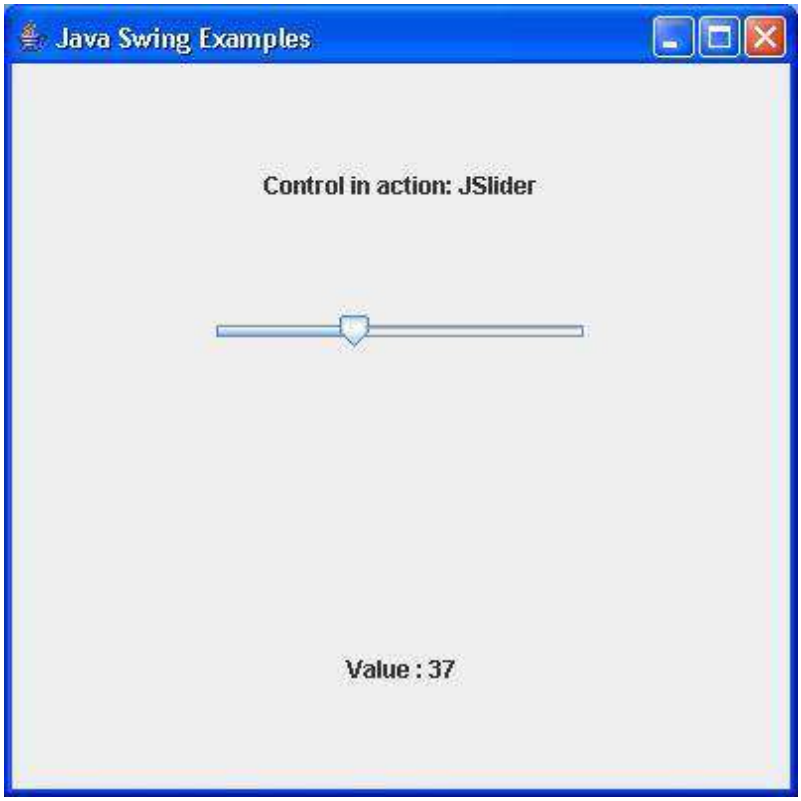
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.



JSpinner Class

Introduction

The class **JSpinner** is a component which lets the user select a number or an object value from an ordered sequence using an input field.

Class Declaration

Following is the declaration for **javax.swing.JSpinner** class:

```
public class JSpinner
   extends JComponent
      implements Accessible
```

Class Constructors

Sr.No.	Constructor & Description
1	**JSpinner()** Constructs a spinner with an Integer SpinnerNumberModel with initial value 0 and no minimum or maximum limits.
2	**JSpinner(SpinnerModel model)** Constructs a complete spinner with a pair of next/previous buttons and an editor for the SpinnerModel.

Class Methods

Sr.No.	Method & Description
1	**void addChangeListener(ChangeListener listener)** Adds a listener to the list who is notified each time a change to the model occurs.
2	**void commitEdit()** Commits the currently edited value to the SpinnerModel.
3	**protected JComponent createEditor(SpinnerModel model)** This method is called by the constructors to create the JComponent that displays the current value of the sequence.
4	**protected void fireStateChanged()** Sends a ChangeEvent, whose source is this JSpinner, to each ChangeListener.
5	

	AccessibleContext getAccessibleContext() Gets the AccessibleContext for the JSpinner.
6	**ChangeListener[] getChangeListeners()** Returns an array of all the ChangeListeners added to this JSpinner with addChangeListener().
7	**JComponent getEditor()** Returns the component that displays and potentially changes the model's value.
8	**SpinnerModel getModel()** Returns the SpinnerModel that defines this spinners sequence of values.
9	**Object getNextValue()** Returns the object in the sequence that comes after the object returned by getValue().
10	**Object getPreviousValue()** Returns the object in the sequence that comes before the object returned by getValue().
11	**SpinnerUI getUI()** Returns the look and feel (L&F) object that renders this component.
12	**String getUIClassID()** Returns the suffix used to construct the name of the look and feel (L&F) class used to render this component.
13	**Object getValue()** Returns the current value of the model, typically this value is displayed by the editor.
14	**void removeChangeListener(ChangeListener listener)** Removes a ChangeListener from this spinner.
15	**void setEditor(JComponent editor)** Changes the JComponent that displays the current value of the SpinnerModel.

16	**void setModel(SpinnerModel model)** Changes the model that represents the value of this spinner.
17	**void setUI(SpinnerUI ui)** Sets the look and feel (L&F) object that renders this component.
18	**void setValue(Object value)** Changes the current value of the model, typically this value is displayed by the editor.
19	**void updateUI()** Resets the UI property with the value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JSpinner Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

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

```
public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new SwingControlDemo();
      swingControlDemo.showSpinnerDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());
```

```
      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showSpinnerDemo(){
      headerLabel.setText("Control in action: JSpinner");
      SpinnerModel spinnerModel =
         new SpinnerNumberModel(10, //initial value
            0, //min
            100, //max
            1);//step
      JSpinner spinner = new JSpinner(spinnerModel);
      spinner.addChangeListener(new ChangeListener() {
         public void stateChanged(ChangeEvent e) {
            statusLabel.setText("Value : "
            + ((JSpinner)e.getSource()).getValue());
         }
      });
      controlPanel.add(spinner);
      mainFrame.setVisible(true);
   }
}
```

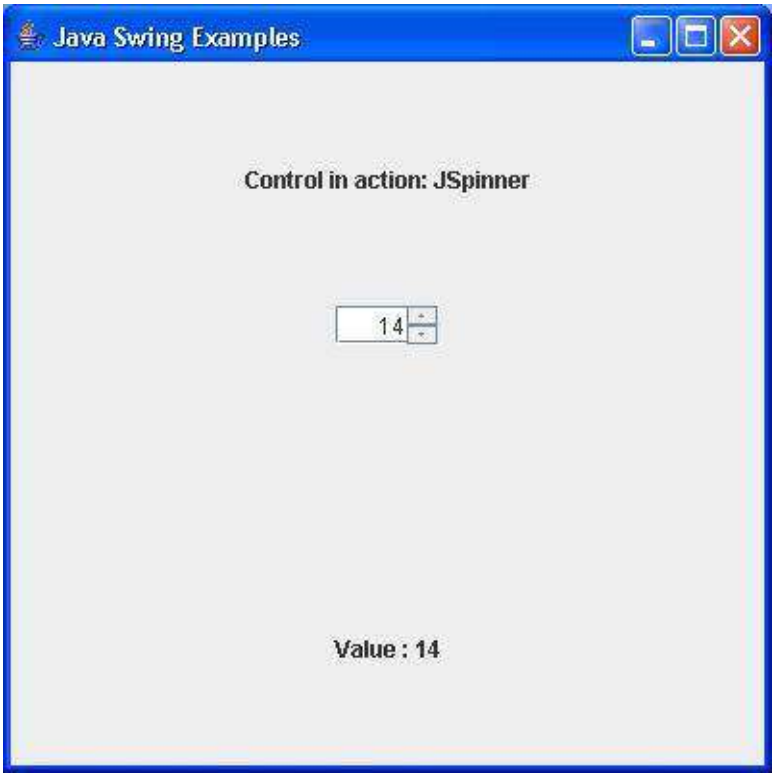
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.



4. Swing – Event Handling

In this chapter, you will learn about Events, its types, and also learn how to handle an event. Example is provided at the end of the chapter for better understanding.

What is an Event?

Change in the state of an object is known as **Event**, i.e., event describes the change in the state of the source. Events are generated as a result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from the list, and scrolling the page are the activities that causes an event to occur.

Types of Event

The events can be broadly classified into two categories:

- **Foreground Events** - These events require direct interaction of the user. They are generated as consequences of a person interacting with the graphical components in the Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page, etc.

- **Background Events** – These events require the interaction of the end user. Operating system interrupts, hardware or software failure, timer expiration, and operation completion are some examples of background events.

What is Event Handling?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has a code which is known as an event handler, that is executed when an event occurs.

Java uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.

The Delegation Event Model has the following key participants.

- **Source** - The source is an object on which the event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide us with classes for the source object.

- **Listener** - It is also known as event handler. The listener is responsible for generating a response to an event. From the point of view of Java implementation, the listener is also an object. The listener waits till it receives an event. Once the event is received, the listener processes the event and then returns.

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

In this model, the listener needs to be registered with the source object so that the listener can receive the event notification. This is an efficient way of handling the event because the event notifications are sent only to those listeners who want to receive them.

Steps Involved in Event Handling

Step 1: The user clicks the button and the event is generated.

Step 2: The object of concerned event class is created automatically and information about the source and the event get populated within the same object.

Step 3: Event object is forwarded to the method of the registered listener class.

Step 4: The method is gets executed and returns.

Points to Remember About the Listener

- In order to design a listener class, you have to develop some listener interfaces. These Listener interfaces forecast some public abstract callback methods, which must be implemented by the listener class.

- If you do not implement any of the predefined interfaces, then your class cannot act as a listener class for a source object.

Callback Methods

These are the methods that are provided by API provider and are defined by the application programmer and invoked by the application developer. Here the callback methods represent an event method. In response to an event, java jre will fire callback method. All such callback methods are provided in listener interfaces.

If a component wants some listener to listen ot its events, the source must register itself to the listener.

Event Handling Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingControlDemo {
```

```
private JFrame mainFrame;
private JLabel headerLabel;
private JLabel statusLabel;
private JPanel controlPanel;

public SwingControlDemo(){
   prepareGUI();
}

public static void main(String[] args){
   SwingControlDemo swingControlDemo = new SwingControlDemo();
   swingControlDemo.showEventDemo();
}

private void prepareGUI(){
   mainFrame = new JFrame("Java SWING Examples");
   mainFrame.setSize(400,400);
   mainFrame.setLayout(new GridLayout(3, 1));

   headerLabel = new JLabel("",JLabel.CENTER );
   statusLabel = new JLabel("",JLabel.CENTER);

   statusLabel.setSize(350,100);
   mainFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent windowEvent){
          System.exit(0);
      }
   });
   controlPanel = new JPanel();
   controlPanel.setLayout(new FlowLayout());

   mainFrame.add(headerLabel);
   mainFrame.add(controlPanel);
   mainFrame.add(statusLabel);
   mainFrame.setVisible(true);
}
```

```
private void showEventDemo(){
   headerLabel.setText("Control in action: Button");

   JButton okButton = new JButton("OK");
   JButton submitButton = new JButton("Submit");
   JButton cancelButton = new JButton("Cancel");

   okButton.setActionCommand("OK");
   submitButton.setActionCommand("Submit");
   cancelButton.setActionCommand("Cancel");

   okButton.addActionListener(new ButtonClickListener());
   submitButton.addActionListener(new ButtonClickListener());
   cancelButton.addActionListener(new ButtonClickListener());

   controlPanel.add(okButton);
   controlPanel.add(submitButton);
   controlPanel.add(cancelButton);

   mainFrame.setVisible(true);
}

private class ButtonClickListener implements ActionListener{
   public void actionPerformed(ActionEvent e) {
      String command = e.getActionCommand();
      if( command.equals( "OK" ))  {
         statusLabel.setText("Ok Button clicked.");
      }
      else if( command.equals( "Submit" ) )  {
         statusLabel.setText("Submit Button clicked.");
      }
      else  {
         statusLabel.setText("Cancel Button clicked.");
      }
   }
}
```

```
}
```

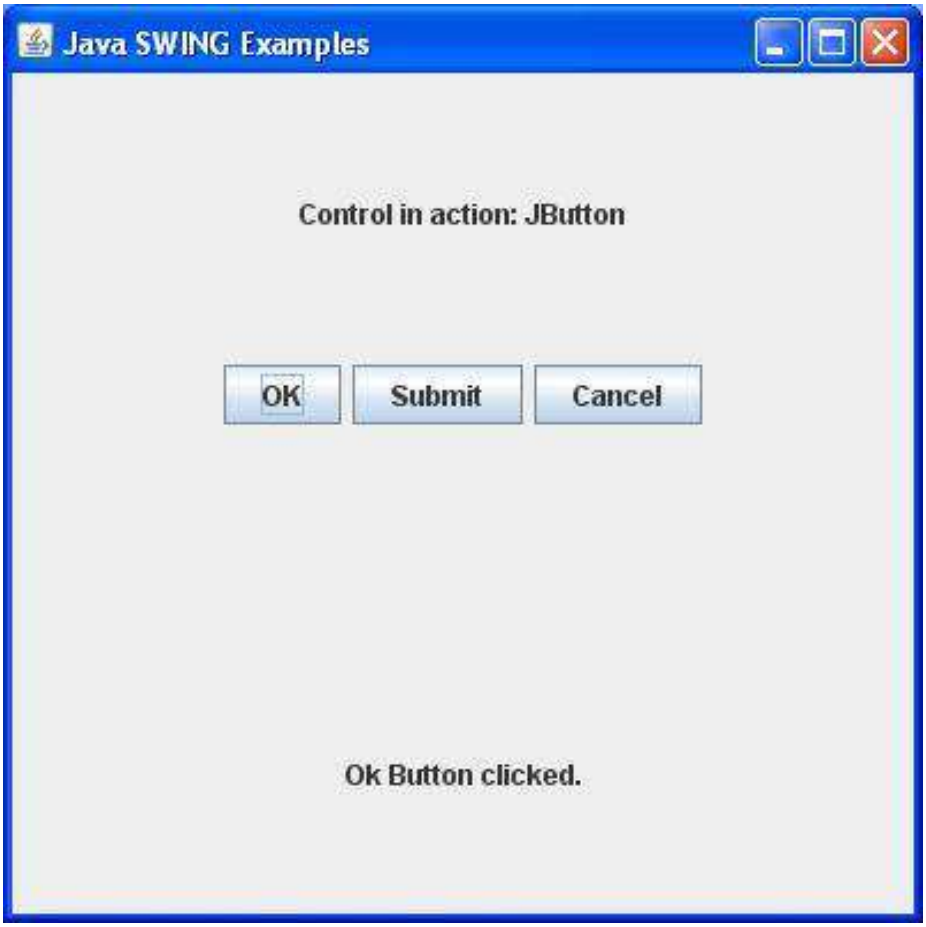
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\AWT>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\AWT>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.



5. Swing – Event Classes

Event classes represent the event. Java provides various Event classes, however, only those which are more frequently used will be discussed.

EventObject Class

It is the root class from which all event state objects shall be derived. All Events are constructed with a reference to the object, the **source**, that is logically deemed to be the object upon which the Event in question initially occurred upon. This class is defined in **java.util** package.

Class Declaration

Following is the declaration for **java.util.EventObject** class:

```
public class EventObject
   extends Object
      implements Serializable
```

Field

Following are the fields for **java.util.EventObject** class:

- **protected Object source** - The object on which the Event initially occurred.

Class Constructors

Sr.No.	Constructor & Description
1	**EventObject(Object source)** Constructs a prototypical Event.

Class Methods

Sr.No.	Method & Description
1	**Object getSource()** The object on which the Event initially occurred.
2	**String toString()** Returns a String representation of this EventObject.

Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

SWING Event Classes

Following is the list of commonly used Event classes.

Sr. No.	Control & Description
1	**AWTEvent** It is the root event class for all SWING events. This class and its subclasses supercede the original **java.awt.Event** class.
2	**ActionEvent** The ActionEvent is generated when the button is clicked or the item of a list is double-clicked.
3	**InputEvent** The InputEvent class is the root event class for all component-level input events.

4	**KeyEvent** On entering the character, the Key event is generated.
5	**MouseEvent** This event indicates a mouse action occurred in a component.
6	**WindowEvent** The object of this class represents the change in the state of a window.
7	**AdjustmentEvent** The object of this class represents the adjustment event emitted by Adjustable objects.
8	**ComponentEvent** The object of this class represents the change in the state of a window.
9	**ContainerEvent** The object of this class represents the change in the state of a window.
10	**MouseMotionEvent** The object of this class represents the change in the state of a window.
11	**PaintEvent** The object of this class represents the change in the state of a window.

AWTEvent Class

It is the root event class for all AWTEvent events. This class and its subclasses supercede the original **java.awt.Event** class. This class is defined in **java.awt** package. This class has a method named **getID()** that can be used to determine the type of event.

Class Declaration

Following is the declaration for **java.awt.AWTEvent** class:

```
public class AWTEvent
   extends EventObject
```

Field

Following are the fields for **java.awt.AWTEvent** class:

- **static int ACTION_FIRST** - The first number in the range of IDs used for action events.
- **static long ACTION_EVENT_MASK** - The event mask for selecting action events.
- **static long ADJUSTMENT_EVENT_MASK** - The event mask for selecting adjustment events.
- **static long COMPONENT_EVENT_MASK** - The event mask for selecting component events.
- **protected boolean consumed** - Controls whether or not the event is sent back down to the peer once the source has processed it - false means it's sent to the peer; true means it's not.
- **static long CONTAINER_EVENT_MASK** - The event mask for selecting container events.
- **static long FOCUS_EVENT_MASK** - The event mask for selecting focus events.
- **static long HIERARCHY_BOUNDS_EVENT_MASK** - The event mask for selecting hierarchy bounds events.
- **static long HIERARCHY_EVENT_MASK** - The event mask for selecting hierarchy events.
- **protected int id** - The event's ID.
- **static long INPUT_METHOD_EVENT_MASK** - The event mask for selecting input method events.
- **static long INVOCATION_EVENT_MASK** - The event mask for selecting invocation events.
- **static long ITEM_EVENT_MASK** - The event mask for selecting item events.
- **static long KEY_EVENT_MASK** - The event mask for selecting key events.
- **static long MOUSE_EVENT_MASK** - The event mask for selecting mouse events.
- **static long MOUSE_MOTION_EVENT_MASK** - The event mask for selecting mouse motion events.
- **static long MOUSE_WHEEL_EVENT_MASK** - The event mask for selecting mouse wheel events.
- **static long PAINT_EVENT_MASK** - The event mask for selecting paint events.

- **static int RESERVED_ID_MAX** - The maximum value for reserved SWING event IDs.
- **static long TEXT_EVENT_MASK** - The event mask for selecting text events.
- **static long WINDOW_EVENT_MASK** - The event mask for selecting window events.
- **static long WINDOW_FOCUS_EVENT_MASK** - The event mask for selecting window focus events.
- **static long WINDOW_STATE_EVENT_MASK** - The event mask for selecting window state events.

Class Constructors

Sr.No.	Constructor & Description
1	**AWTEvent(Event event)** Constructs an AWTEvent object from the parameters of a 1.0-style event.
2	**AWTEvent(java.lang.Object source, int id)** Constructs an AWTEvent object with the specified source object and type.

Class Methods

Sr.No.	Method & Description
1	**protected void consume()** Consumes this event, if this event can be consumed.
2	**int getID()** Returns the event type.

3	**protected boolean isConsumed()** Returns whether this event has been consumed.
4	**java.lang.String paramString()** Returns a string representing the state of this Event.
5	**void setSource(java.lang.Object newSource)** Retargets an event to a new source.
6	**java.lang.String toString()** Returns a String representation of this object.

Methods Inherited

This class inherits methods from the following classes:

- java.util.EventObject
- java.lang.Object

ActionEvent Class

This class is defined in **java.awt.event** package. The ActionEvent is generated when the button is clicked or the item of a list is double-clicked.

Class Declaration

Following is the declaration for **java.awt.event.ActionEvent** class:

```
public class ActionEvent
   extends AWTEvent
```

Field

Following are the fields for **java.awt.event.ActionEvent** class:

- **static int ACTION_FIRST** - The first number in the range of IDs used for action events.
- **static int ACTION_LAST** - The last number in the range of IDs used for action events.
- **static int ACTION_PERFORMED** - This event ID indicates that a meaningful action has occured.
- **static int ALT_MASK** - The alt modifier.

- **static int CTRL_MASK** - The control modifier.
- **static int META_MASK** - The meta modifier.
- **static int SHIFT_MASK** - The shift modifier.

Class Constructors

Sr.No.	Constructor & Description
1	**ActionEvent(java.lang.Object source, int id, java.lang.String command)** Constructs an ActionEvent object.
2	**ActionEvent(java.lang.Object source, int id, java.lang.String command, int modifiers)** Constructs an ActionEvent object with modifier keys.
3	**ActionEvent(java.lang.Object source, int id, java.lang.String command, long when, int modifiers)** Constructs an ActionEvent object with the specified modifier keys and timestamp.

Class Methods

Sr.No.	Method & Description
1	**java.lang.String getActionCommand()** Returns the command string associated with this action.
2	**int getModifiers()** Returns the modifier keys held down during this action event.
3	**long getWhen()** Returns the timestamp of when this event occurred.
4	**java.lang.String paramString()** Returns a parameter string identifying this action event.

Methods Inherited

This class inherits methods from the following classes:

- java.awt.AWTEvent
- java.util.EventObject
- java.lang.Object

InputEvent Class

The InputEvent class is the root event class for all component-level input events. Input events are delivered to the listeners before they are processed normally by the source where they originated. This allows the listeners and component subclasses to "consume" the event so that the source will not process them in their default manner. For example, consuming mousePressed events on a Button component will prevent the Button from being activated.

Class Declaration

Following is the declaration for **java.awt.event.InputEvent** class:

```
public abstract class InputEvent
   extends ComponentEvent
```

Field

Following are the fields for **java.awt.event.InputEvent** class:

- **static int ALT_DOWN_MASK** - The Alt key extended modifier constant.
- **static int ALT_GRAPH_DOWN_MASK** - The AltGraph key extended modifier constant.
- **static int ALT_GRAPH_MASK** - The AltGraph key modifier constant.
- **static int ALT_MASK** - The Alt key modifier constant.
- **static int BUTTON1_DOWN_MASK** - The Mouse Button1 extended modifier constant.
- **static int BUTTON1_MASK** - The Mouse Button1 modifier constant.
- **static int BUTTON2_DOWN_MASK** - The Mouse Button2 extended modifier constant.
- **static int BUTTON2_MASK** - The Mouse Button2 modifier constant.
- **static int BUTTON3_DOWN_MASK** - The Mouse Button3 extended modifier constant.

- **static int BUTTON3_MASK** - The Mouse Button3 modifier constant.
- **static int CTRL_DOWN_MASK** - The Control key extended modifier constant.
- **static int CTRL_MASK** - The Control key modifier constant.
- **static int META_DOWN_MASK** - The Meta key extended modifier constant.
- **static int META_MASK** - The Meta key modifier constant.
- **static int SHIFT_DOWN_MASK** - The Shift key extended modifier constant.
- **static int SHIFT_MASK** - The Shift key modifier constant.

Class Methods

Sr.No.	Method & Description
1	**void consume()** Consumes this event so that it will not be processed in the default manner by the source which originated it.
2	**int getModifiers()** Returns the modifier mask for this event.
3	**int getModifiersEx()** Returns the extended modifier mask for this event.
4	**static String getModifiersExText(int modifiers)** Returns a String describing the extended modifier keys and mouse buttons, such as "Shift", "Button1", or "Ctrl+Shift".
5	**long getWhen()** Returns the timestamp, when this event occurred.

6	**boolean isAltDown()** Returns whether or not the Alt modifier is down on this event.
7	**boolean isAltGraphDown()** Returns whether or not the AltGraph modifier is down on this event.
8	**boolean isConsumed()** Returns whether or not this event has been consumed.
9	**boolean isControlDown()** Returns whether or not the Control modifier is down on this event.
10	**boolean isMetaDown()** Returns whether or not the Meta modifier is down on this event.
11	**boolean isShiftDown()** Returns whether or not the Shift modifier is down on this event.

Methods Inherited

This class inherits methods from the following classes:

- java.awt.event.ComponentEvent
- java.awt.AWTEvent
- java.util.EventObject
- java.lang.Object

KeyEvent Class

On entering the character, the Key event is generated. There are three types of key events which are represented by the integer constants.

- KEY_PRESSED
- KEY_RELASED
- KEY_TYPED

Class Declaration

Following is the declaration for **java.awt.event.KeyEvent** class:

```
public class KeyEvent
   extends InputEvent
```

Field

Following are the fields for **java.awt.InputEvent** class:

- **static char CHAR_UNDEFINED** - KEY_PRESSED and KEY_RELEASED events which do not map to a valid Unicode character use this for the keyChar value.
- **static int KEY_FIRST** - The first number in the range of IDs used for key events.
- **static int KEY_LAST** - The last number in the range of IDs used for key events.
- **static int KEY_LOCATION_LEFT** - A constant indicating that the key pressed or released is in the left key location (there is more than one possible location for this key).
- **static int KEY_LOCATION_NUMPAD** - A constant indicating that the key event originated on the numeric keypad or with a virtual key corresponding to the numeric keypad.
- **static int KEY_LOCATION_RIGHT** - A constant indicating that the key pressed or released is in the right key location (there is more than one possible location for this key).
- **static int KEY_LOCATION_STANDARD** - A constant indicating that the key pressed or released is not distinguished as the left or right version of a key, and did not originate on the numeric keypad (or did not originate with a virtual key corresponding to the numeric keypad).
- **static int KEY_LOCATION_UNKNOWN** - A constant indicating that the keyLocation is indeterminate or not relevant.
- **static int KEY_PRESSED** - The "key pressed" event.
- **static int KEY_RELEASED** - The "key released" event.
- **static int KEY_TYPED** - The "key typed" event.
- **static int VK_0** - VK_0 thru VK_9 are the same as ASCII '0' thru '9' (0x30 - 0x39)
- **static int VK_1**
- **static int VK_2**
- **static int VK_3**
- **static int VK_4**
- **static int VK_5**

- **static int VK_6**
- **static int VK_7**
- **static int VK_8**
- **static int VK_9**
- **static int VK_A** - VK_A thru VK_Z are the same as ASCII 'A' thru 'Z' (0x41 - 0x5A)
- **static int VK_ACCEPT** - Constant for the Accept or Commit function key.
- **static int VK_ADD**
- **static int VK_AGAIN**
- **static int VK_ALL_CANDIDATES** - Constant for the All Candidates function key.
- **static int VK_ALPHANUMERIC** - Constant for the Alphanumeric function key.
- **static int VK_ALT**
- **static int VK_ALT_GRAPH** - Constant for the AltGraph function key.
- **static int VK_AMPERSAND**
- **static int VK_ASTERISK**
- **static int VK_AT** - Constant for the "@" key.
- **static int VK_B**
- **static int VK_BACK_QUOTE**
- **static int VK_BACK_SLASH** - Constant for the back slash key, "\"
- **static int VK_BACK_SPACE**
- **static int VK_BEGIN** - Constant for the Begin key.
- **static int VK_BRACELEFT**
- **static int VK_BRACERIGHT**
- **static int VK_C**
- **static int VK_CANCEL**
- **static int VK_CAPS_LOCK**
- **static int VK_CIRCUMFLEX** - Constant for the "^" key.
- **static int VK_CLEAR**
- **static int VK_CLOSE_BRACKET** - Constant for the close bracket key, "]"
- **static int VK_CODE_INPUT** - Constant for the Code Input function key.
- **static int VK_COLON** - Constant for the ":" key.

- **static int VK_COMMA** - Constant for the comma key, ","
- **static int VK_COMPOSE** - Constant for the Compose function key.
- **static int VK_CONTEXT_MENU** - Constant for the Microsoft Windows Context Menu key.
- **static int VK_CONTROL**
- **static int VK_CONVERT** - Constant for the Convert function key.
- **static int VK_COPY**
- **static int VK_CUT**
- **static int VK_D**
- **static int VK_DEAD_ABOVEDOT**
- **static int VK_DEAD_ABOVERING**
- **static int VK_DEAD_ACUTE**
- **static int VK_DEAD_BREVE**
- **static int VK_DEAD_CARON**
- **static int VK_DEAD_CEDILLA**
- **static int VK_DEAD_CIRCUMFLEX**
- **static int VK_DEAD_DIAERESIS**
- **static int VK_DEAD_DOUBLEACUTE**
- **static int VK_DEAD_GRAVE**
- **static int VK_DEAD_IOTA**
- **static int VK_DEAD_MACRON**
- **static int VK_DEAD_OGONEK**
- **static int VK_DEAD_SEMIVOICED_SOUND**
- **static int VK_DEAD_TILDE**
- **static int VK_DEAD_VOICED_SOUND**
- **static int VK_DECIMAL**
- **static int VK_DELETE**
- **static int VK_DIVIDE**
- **static int VK_DOLLAR** - Constant for the "$" key.
- **static int VK_DOWN** - Constant for the non-numpad down arrow key.
- **static int VK_E**
- **static int VK_END**
- **static int VK_ENTER**
- **static int VK_EQUALS** - Constant for the equals key, "="

- **static int VK_ESCAPE**
- **static int VK_EURO_SIGN** - Constant for the Euro currency sign key.
- **static int VK_EXCLAMATION_MARK** - Constant for the "!" key.
- **static int VK_F**
- **static int VK_F1** - Constant for the F1 function key.
- **static int VK_F10** - Constant for the F10 function key.
- **static int VK_F11** - Constant for the F11 function key.
- **static int VK_F12** - Constant for the F12 function key.
- **static int VK_F13** - Constant for the F13 function key.
- **static int VK_F14** - Constant for the F14 function key.
- **static int VK_F15** - Constant for the F15 function key.
- **static int VK_F16** - Constant for the F16 function key.
- **static int VK_F17** - Constant for the F17 function key.
- **static int VK_F18** - Constant for the F18 function key.
- **static int VK_F19** - Constant for the F19 function key.
- **static int VK_F2** - Constant for the F2 function key.
- **static int VK_F20** - Constant for the F20 function key.
- **static int VK_F21** - Constant for the F21 function key.
- **static int VK_F22** - Constant for the F22 function key.
- **static int VK_F23** - Constant for the F23 function key.
- **static int VK_F24** - Constant for the F24 function key.
- **static int VK_F3** - Constant for the F3 function key.
- **static int VK_F4** - Constant for the F4 function key.
- **static int VK_F5** - Constant for the F5 function key.
- **static int VK_F6** - Constant for the F6 function key.
- **static int VK_F7** - Constant for the F7 function key.

- **static int VK_F8** - Constant for the F8 function key.
- **static int VK_F9** - Constant for the F9 function key.
- **static int VK_FINAL**
- **static int VK_FIND**
- **static int VK_FULL_WIDTH** - Constant for the Full-Width Characters function key.
- **static int VK_G**
- **static int VK_GREATER**
- **static int VK_H**
- **static int VK_HALF_WIDTH** - Constant for the Half-Width Characters function key.
- **static int VK_HELP**
- **static int VK_HIRAGANA** - Constant for the Hiragana function key.
- **static int VK_HOME**
- **static int VK_I**
- **static int VK_INPUT_METHOD_ON_OFF** - Constant for the input method on/off key.
- **static int VK_INSERT**
- **static int VK_INVERTED_EXCLAMATION_MARK** - Constant for the inverted exclamation mark key.
- **static int VK_J**
- **static int VK_JAPANESE_HIRAGANA** - Constant for the Japanese-Hiragana function key.
- **static int VK_JAPANESE_KATAKANA** - Constant for the Japanese-Katakana function key.
- **static int VK_JAPANESE_ROMAN** - Constant for the Japanese-Roman function key.
- **static int VK_K**
- **static int VK_KANA**
- **static int VK_KANA_LOCK** - Constant for the locking Kana function key.
- **static int VK_KANJI**
- **static int VK_KATAKANA** - Constant for the Katakana function key.
- **static int VK_KP_DOWN** - Constant for the numeric keypad down arrow key.

- **static int VK_KP_LEFT** - Constant for the numeric keypad left arrow key.
- **static int VK_KP_RIGHT** - Constant for the numeric keypad right arrow key.
- **static int VK_KP_UP** - Constant for the numeric keypad up arrow key.
- **static int VK_L**
- **static int VK_LEFT** - Constant for the non-numpad left arrow key.
- **static int VK_LEFT_PARENTHESIS** - Constant for the "(" key.
- **static int VK_LESS**
- **static int VK_M**
- **static int VK_META**
- **static int VK_MINUS** - Constant for the minus key, "-"
- **static int VK_MODECHANGE**
- **static int VK_MULTIPLY**
- **static int VK_N**
- **static int VK_NONCONVERT** - Constant for the Don't Convert function key.
- **static int VK_NUM_LOCK**
- **static int VK_NUMBER_SIGN** - Constant for the "#" key.
- **static int VK_NUMPAD0**
- **static int VK_NUMPAD1**
- **static int VK_NUMPAD2**
- **static int VK_NUMPAD3**
- **static int VK_NUMPAD4**
- **static int VK_NUMPAD5**
- **static int VK_NUMPAD6**
- **static int VK_NUMPAD7**
- **static int VK_NUMPAD8**
- **static int VK_NUMPAD9**
- **static int VK_O**
- **static int VK_OPEN_BRACKET** - Constant for the open bracket key, "["
- **static int VK_P**
- **static int VK_PAGE_DOWN**
- **static int VK_PAGE_UP**
- **static int VK_PASTE**

- **static int VK_PAUSE**
- **static int VK_PERIOD** - Constant for the period key, "."
- **static int VK_PLUS** - Constant for the "+" key.
- **static int VK_PREVIOUS_CANDIDATE** - Constant for the Previous Candidate function key.
- **static int VK_PRINTSCREEN**
- **static int VK_PROPS**
- **static int VK_Q**
- **static int VK_QUOTE**
- **static int VK_QUOTEDBL**
- **static int VK_R**
- **static int VK_RIGHT** - Constant for the non-numpad right arrow key.
- **static int VK_RIGHT_PARENTHESIS** - Constant for the ")" key.
- **static int VK_ROMAN_CHARACTERS** - Constant for the Roman Characters function key.
- **static int VK_S**
- **static int VK_SCROLL_LOCK**
- **static int VK_SEMICOLON** - Constant for the semicolon key, ";"
- **static int VK_SEPARATER** - This constant is obsolete, and is included only for backwards compatibility.
- **static int VK_SEPARATOR** - Constant for the Numpad Separator key.
- **static int VK_SHIFT**
- **static int VK_SLASH** - Constant for the forward slash key, "/"
- **static int VK_SPACE**
- **static int VK_STOP**
- **static int VK_SUBTRACT**
- **static int VK_T**
- **static int VK_TAB**
- **static int VK_U**
- **static int VK_UNDEFINED** - This value is used to indicate that the keyCode is unknown.
- **static int VK_UNDERSCORE** - Constant for the "_" key.
- **static int VK_UNDO**

- **static int VK_UP** - Constant for the non-numpad up arrow key.
- **static int VK_V**
- **static int VK_W**
- **static int VK_WINDOWS** - Constant for the Microsoft Windows "Windows" key.
- **static int VK_X**
- **static int VK_Y**
- **static int VK_Z**

Class Constructors

Sr.No.	Constructor & Description
1	**KeyEvent(Component source, int id, long when, int modifiers, int keyCode)** Deprecated. as of JDK1.1
2	**KeyEvent(Component source, int id, long when, int modifiers, int keyCode, char keyChar)** Constructs a KeyEvent object.
3	**KeyEvent(Component source, int id, long when, int modifiers, int keyCode, char keyChar, int keyLocation)**

Class Methods

Sr.No.	Method & Description
1	**char getKeyChar()** Returns the character associated with the key in this event.
2	**int getKeyCode()** Returns the integer keyCode associated with the key in this event.
3	**int getKeyLocation()**

	Returns the location of the key that originated this key event.
4	**static String getKeyModifiersText(int modifiers)** Returns a String describing the modifier key(s), such as "Shift", or "Ctrl+Shift".
5	**static String getKeyText(int keyCode)** Returns a String describing the keyCode, such as "HOME", "F1" or "A".
6	**boolean isActionKey()** Returns whether the key in this event is an "action" key.
7	**String paramString()** Returns a parameter string identifying this event.
8	**void setKeyChar(char keyChar)** Set the keyChar value to indicate a logical character.
9	**void setKeyCode(int keyCode)** Set the keyCode value to indicate a physical key.
10	**void setModifiers(int modifiers)** Deprecated. as of JDK1.1.4

Methods Inherited

This class inherits methods from the following classes:

- java.awt.event.InputEvent
- java.awt.event.ComponentEvent
- java.awt.AWTEvent
- java.util.EventObject
- java.lang.Object

MouseEvent Class

This event indicates that a mouse action occurred in a component. This low-level event is generated by a component object for Mouse Events and Mouse motion events. This event is generated when -

- A mouse button is pressed.

- A mouse button is released.
- A mouse button is clicked (pressed and released).
- A mouse cursor enters the unobscured part of component's geometry.
- A mouse cursor exits the unobscured part of component's geometry.
- A mouse is moved.
- A mouse is dragged.

Class Declaration

Following is the declaration for **java.awt.event.MouseEvent** class:

```
public class MouseEvent
   extends InputEvent
```

Field

Following are the fields for **java.awt.event.MouseEvent** class:

- **static int BUTTON1** - Indicates mouse button #1; used by getButton()
- **static int BUTTON2** - Indicates mouse button #2; used by getButton()
- **static int BUTTON3** - Indicates mouse button #3; used by getButton()
- **static int MOUSE_CLICKED** - The "mouse clicked" event
- **static int MOUSE_DRAGGED** - The "mouse dragged" event
- **static int MOUSE_ENTERED** - The "mouse entered" event
- **static int MOUSE_EXITED** - The "mouse exited" event
- **static int MOUSE_FIRST** - The first number in the range of IDs used for mouse events.
- **static int MOUSE_LAST** - The last number in the range of IDs used for mouse events.
- **static int MOUSE_MOVED** - The "mouse moved" event.
- **static int MOUSE_PRESSED** - The "mouse pressed" event.
- **static int MOUSE_RELEASED** - The "mouse released" event.
- **static int MOUSE_WHEEL** - The "mouse wheel" event.
- **static int NOBUTTON** - Indicates no mouse buttons; used by getButton()
- **static int VK_WINDOWS** - Constant for the Microsoft Windows "Windows" key.

Class Constructors

Sr.No.	Constructor & Description
1	**MouseEvent(Component source, int id, long when, int modifiers, int x, int y, int clickCount, boolean popupTrigger)** Constructs a MouseEvent object with the specified source component, type, modifiers, coordinates, and click count.
2	**MouseEvent(Component source, int id, long when, int modifiers, int x, int y, int clickCount, boolean popupTrigger, int button)** Constructs a MouseEvent object with the specified source component, type, modifiers, coordinates, and click count.
3	**MouseEvent(Component source, int id, long when, int modifiers, int x, int y, int xAbs, int yAbs, int clickCount, boolean popupTrigger, int button)** Constructs a MouseEvent object with the specified source component, type, modifiers, coordinates, absolute coordinates, and click count.

Class Methods

Sr.No.	Method & Description
1	**int getButton()** Returns which, if any, of the mouse buttons has changed state.
2	**int getClickCount()** Returns the number of mouse clicks associated with this event.
3	**Point getLocationOnScreen()** Returns the absolute **x, y** position of the event.
4	**static String getMouseModifiersText(int modifiers)** Returns a String describing the modifier keys and mouse buttons that were down during the event, such as "Shift", or "Ctrl+Shift".
5	**Point getPoint()**

	Returns the **x,y** position of the event relative to the source component.
6	**int getX()** Returns the horizontal **x** position of the event relative to the source component.
7	**int getXOnScreen()** Returns the absolute horizontal **x** position of the event.
8	**int getY()** Returns the vertical **y** position of the event relative to the source component.
9	**int getYOnScreen()** Returns the absolute vertical **y** position of the event.
10	**boolean isPopupTrigger()** Returns whether or not this mouse event is the popup menu trigger event for the platform.
11	**String paramString()** Returns a parameter string identifying this event.
12	**void translatePoint(int x, int y)** Translates the event's coordinates to a new position by adding specified **x** (horizontal) and **y** (vertical) offsets.

Methods Inherited

This class inherits methods from the following classes:

- java.awt.event.InputEvent
- java.awt.event.ComponentEvent
- java.awt.AWTEvent
- java.util.EventObject
- java.lang.Object

WindowEvent Class

The object of this class represents the change in state of a window. This low-level event is generated by a Window object when it is opened, closed, activated, deactivated, iconified, or deiconified, or when the focus is transfered into or out of the Window.

Class Declaration

Following is the declaration for **java.awt.event.WindowEvent** class:

```
public class WindowEvent
   extends ComponentEvent
```

Field

Following are the fields for **java.awt.event.WindowEvent** class:

- **static int WINDOW_ACTIVATED** - The window-activated event type.
- **static int WINDOW_CLOSED** - The window closed event.
- **static int WINDOW_CLOSING** - The "window is closing" event.
- **static int WINDOW_DEACTIVATED** - The window-deactivated event type.
- **static int WINDOW_DEICONIFIED** - The window deiconified event type.
- **static int WINDOW_FIRST** - The first number in the range of IDs used for window events.
- **static int WINDOW_GAINED_FOCUS** - The window-gained-focus event type.
- **static int WINDOW_ICONIFIED** - The window iconified event.
- **static int WINDOW_LAST** - The last number in the range of IDs used for window events.
- **static int WINDOW_LOST_FOCUS** - The window-lost-focus event type.
- **static int WINDOW_OPENED** - The window opened event.
- **static int WINDOW_STATE_CHANGED** - The window-state-changed event type.

Class Constructors

Sr.No.	Constructor & Description
1	**WindowEvent(Window source, int id)** Constructs a WindowEvent object.
2	**WindowEvent(Window source, int id, int oldState, int newState)** Constructs a WindowEvent object with the specified previous and new window states.

3	**WindowEvent(Window source, int id, Window opposite)** Constructs a WindowEvent object with the specified opposite Window.
4	**WindowEvent(Window source, int id, Window opposite, int oldState, int newState)** Constructs a WindowEvent object.

Class Methods

Sr. No.	Method & Description
1	**int getNewState()** For WINDOW_STATE_CHANGED events returns the new state of the window.
2	**int getOldState()** For WINDOW_STATE_CHANGED events returns the previous state of the window.
3	**Window getOppositeWindow()** Returns the other Window involved in this focus or activation change.
4	**Window getWindow()** Returns the originator of the event.
5	**String paramString()** Returns a parameter string identifying this event.

Methods Inherited

This class inherits methods from the following classes:

- java.awt.event.ComponentEvent
- java.awt.AWTEvent
- java.util.EventObject
- java.lang.Object

AdjustmentEvent Class

Introduction

The Class **AdjustmentEvent** represents adjustment event emitted by Adjustable objects.

Class Declaration

Following is the declaration for **java.awt.event.AdjustmentEvent** class:

```
public class AdjustmentEvent
   extends AWTEvent
```

Field

Following are the fields for **java.awt.Component** class:

- **static int ADJUSTMENT_FIRST** - Marks the first integer ID for the range of adjustment event IDs.
- **static int ADJUSTMENT_LAST** - Marks the last integer ID for the range of adjustment event IDs.
- **static int ADJUSTMENT_VALUE_CHANGED** - The adjustment value changed event.
- **static int BLOCK_DECREMENT** - The block decrement adjustment type.
- **static int BLOCK_INCREMENT** - The block increment adjustment type.
- **static int TRACK** - The absolute tracking adjustment type.
- **static int UNIT_DECREMENT** - The unit decrement adjustment type.
- **static int UNIT_INCREMENT** - The unit increment adjustment type.

Class Constructors

Sr.No.	Constructor & Description
1	**AdjustmentEvent(Adjustable source, int id, int type, int value)** Constructs an AdjustmentEvent object with the specified Adjustable source, event type, adjustment type, and value.
2	**AdjustmentEvent(Adjustable source, int id, int type, int value, boolean isAdjusting)**

	Constructs an AdjustmentEvent object with the specified Adjustable source, event type, adjustment type, and value.

Class Methods

Sr.No.	Method & Description
1	**Adjustable getAdjustable()** Returns the Adjustable object where this event originated.
2	**int getAdjustmentType()** Returns the type of adjustment which caused the value changed event.
3	**int getValue()** Returns the current value in the adjustment event.
4	**boolean getValueIsAdjusting()** Returns true if this is one of the multiple adjustment events.
5	**String paramString()** Returns a string representing the state of this Event.

Methods Inherited

This interface inherits methods from the following classes:

- java.awt.AWTEvent
- java.util.EventObject
- java.lang.Object

ComponentEvent Class

Introduction

The Class **ComponentEvent** represents that a component has moved, changed size, or changed visibility.

Class Declaration

Following is the declaration for **java.awt.event.ComponentEvent** class:

```
public class ComponentEvent
   extends AWTEvent
```

Field

Following are the fields for **java.awt.Component** class:

- **static int COMPONENT_FIRST** - The first number in the range of IDs used for component events.
- **static int COMPONENT_HIDDEN** - This event indicates that the component was rendered invisible.
- **static int COMPONENT_LAST**-- The last number in the range of IDs used for component events.
- **static int COMPONENT_MOVED** - This event indicates that the component's position has changed.
- **static int COMPONENT_RESIZED** - This event indicates that the component's size has changed.
- **static int COMPONENT_SHOWN** - This event indicates that the component was made visible.

Class Constructors

Sr.No.	Constructor & Description
1	**ComponentEvent(Component source, int id)** Constructs a ComponentEvent object.

Class Methods

Sr.No.	Method & Description
1	**Component getComponent()** Returns the originator of the event.
2	**String paramString()** Returns a parameter string identifying this event.

Methods Inherited

This interface inherits methods from the following classes:

- java.awt.AWTEvent
- java.util.EventObject
- java.lang.Object

ContainerEvent Class

Introduction

The Class **ContainerEvent** represents that a container's contents changed because a component was added or removed.

Class Declaration

Following is the declaration for **java.awt.event.ContainerEvent** class:

```
public class ContainerEvent
   extends ComponentEvent
```

Field

Following are the fields for **java.awt.Component** class:

- **static int COMPONENT_ADDED** - This event indicates that a component was added to the container.
- **static int COMPONENT_REMOVED** - This event indicates that a component was removed from the container.
- **static int CONTAINER_FIRST** - The first number in the range of IDs used for container events.
- **static int CONTAINER_LAST** - The last number in the range of IDs used for container events.

Class Constructors

Sr.No.	Constructor & Description
1	**ContainerEvent(Component source, int id, Component child)** Constructs a ContainerEvent object.

Class Methods

Sr.No.	Method & Description
1	**Component getChild()** Returns the component that was affected by the event.
2	**Container getContainer()** Returns the originator of the event.
3	**String paramString()** Returns a parameter string identifying this event.

Methods Inherited

This class inherits methods from the following classes:

- java.awt.ComponentEvent
- java.awt.AWTEvent
- java.util.EventObject
- java.lang.Object

MouseMotionEvent Class

Introduction

The interface **MouseMotionEvent** indicates a mouse action occurred in a component. This low-level event is generated by a component object when a mouse is dragged or moved.

Class Declaration

Following is the declaration for **java.awt.event.MouseMotionEvent** Class:

```
public class MouseMotionEvent
   extends InputEvent
```

Interface Methods

Sr.No.	Method & Description
1	**void mouseDragged(MouseEvent e)**

	Invoked when a mouse button is pressed on a component and then dragged.
2	**void mouseMoved(MouseEvent e)** Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.

Methods Inherited

This interface inherits methods from the following classes:

- java.awt.event.InputEvent
- java.awt.event.ComponentEvent
- java.awt.AWTEvent
- java.util.EventObject
- java.lang.Object

PaintEvent Class

Introduction

The Class **PaintEvent** is used to ensure that paint/update method calls are serialized along with the other events delivered from the event queue.

Class Declaration

Following is the declaration for **java.awt.event.PaintEvent** class:

```
public class PaintEvent
   extends ComponentEvent
```

Field

Following are the fields for **java.awt.Component** class:

- **static int PAINT** - The paint event type.
- **static int PAINT_FIRST** - Marks the first integer ID for the range of paint event IDs.
- **static int PAINT_LAST** - Marks the last integer ID for the range of paint event IDs.
- **static int UPDATE** - The update event type.

Class Constructors

Sr.No.	Constructor & Description
1	**PaintEvent(Component source, int id, Rectangle updateRect)** Constructs a PaintEvent object with the specified source component and type.

Class Methods

Sr.No.	Method & Description
1	**Rectangle getUpdateRect()** Returns the rectangle representing the area which needs to be repainted in response to this event.
2	**String paramString()** Returns a parameter string identifying this event.
3	**void setUpdateRect(Rectangle updateRect)** Sets the rectangle representing the area which needs to be repainted in response to this event.

Methods Inherited

This class inherits methods from the following classes:

- java.awt.ComponentEvent
- java.awt.AWTEvent
- java.util.EventObject
- java.lang.Object

6. Swing – Event Listeners

Event listeners represent the interfaces responsible to handle events. Java provides various Event listener classes, however, only those which are more frequently used will be discussed. Every method of an event listener method has a single argument as an object which is the subclass of EventObject class. For example, mouse event listener methods will accept instance of MouseEvent, where MouseEvent derives from EventObject.

EventListener Interface

It is a marker interface which every listener interface has to extend. This class is defined in **java.util** package.

Class Declaration

Following is the declaration for java.util.EventListener interface:

```
public interface EventListener
```

SWING Event Listener Interfaces

Following is the list of commonly used event listeners.

Sr. No.	Control & Description
1	**ActionListener** This interface is used for receiving the action events.
2	**ComponentListener** This interface is used for receiving the component events.
3	**ItemListener** This interface is used for receiving the item events.
4	**KeyListener** This interface is used for receiving the key events.

5	**MouseListener** This interface is used for receiving the mouse events.
6	**WindowListener** This interface is used for receiving the window events.
7	**AdjustmentListener** This interface is used for receiving the adjustment events.
8	**ContainerListener** This interface is used for receiving the container events.
9	**MouseMotionListener** This interface is used for receiving the mouse motion events.
10	**FocusListener** This interface is used for receiving the focus events.

ActionListener Interface

The class which processes the ActionEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using the **addActionListener()** method. When the action event occurs, that object's actionPerformed method is invoked.

Interface Declaration

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

```
public interface ActionListener
   extends EventListener
```

Interface Methods

Sr.No.	Method & Description
1	**void actionPerformed(ActionEvent e)** Invoked when an action occurs.

Methods Inherited

This interface inherits methods from the following interfaces:

- java.awt.EventListener

ActionListener Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingListenerDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingListenerDemo  swingListenerDemo = new SwingListenerDemo();
      swingListenerDemo.showActionListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);
```

```
      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
             System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showActionListenerDemo(){
      headerLabel.setText("Listener in action: ActionListener");

      JPanel panel = new JPanel();
      panel.setBackground(Color.magenta);

      JButton okButton = new JButton("OK");

      okButton.addActionListener(new CustomActionListener());
      panel.add(okButton);
      controlPanel.add(panel);
      mainFrame.setVisible(true);
   }

   class CustomActionListener implements ActionListener{
      public void actionPerformed(ActionEvent e) {
          statusLabel.setText("Ok Button Clicked.");
      }
   }
}
```

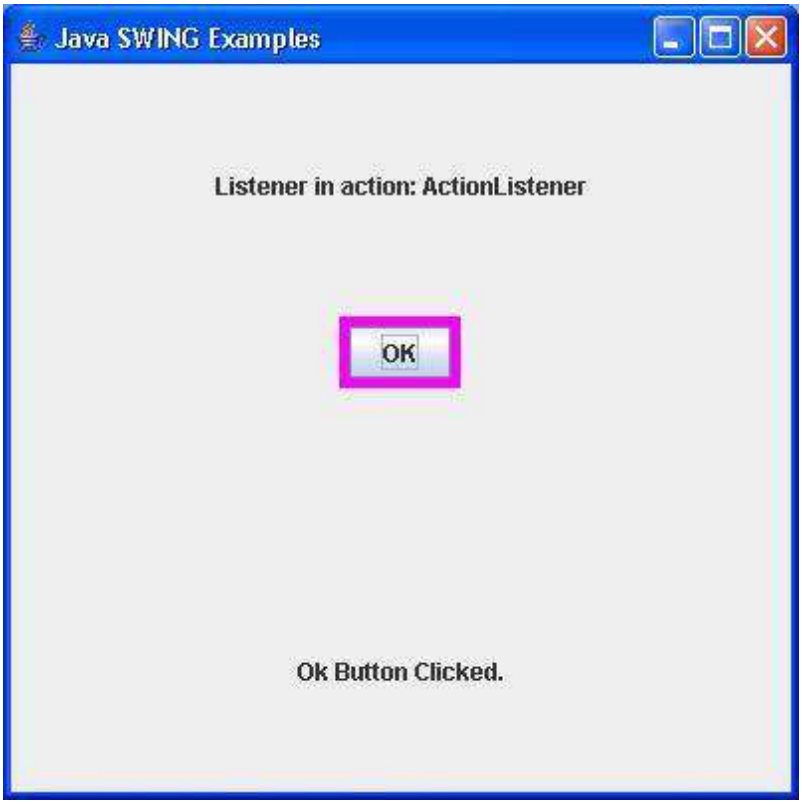
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingListenerDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingListenerDemo
```

Verify the following output.



ComponentListener Interface

The class which processes the ComponentEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using the **addComponentListener()** method. Component event are raised for information only.

Interface Declaration

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

```
public interface ComponentListener

   extends EventListener
```

Interface Methods

Sr.No.	Method & Description
1	**void componentHidden(ComponentEvent e)** Invoked when the component has been made invisible.
2	**void componentMoved(ComponentEvent e)** Invoked when the component's position changes.
3	**void componentResized(ComponentEvent e)** Invoked when the component's size changes.
4	**void componentShown(ComponentEvent e)** Invoked when the component has been made visible.

Methods Inherited

This interface inherits methods from the following interface:

- java.awt.EventListener

ComponentListener Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingListenerDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingListenerDemo(){
```

```
      prepareGUI();
   }

   public static void main(String[] args){
      SwingListenerDemo  swingListenerDemo = new SwingListenerDemo();
      swingListenerDemo.showComponentListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
             System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showComponentListenerDemo(){
      headerLabel.setText("Listener in action: ComponentListener");

      JPanel panel = new JPanel();
      panel.setBackground(Color.magenta);
```

```
      JLabel msglabel =
      new JLabel("Welcome to TutorialsPoint SWING Tutorial."
         ,JLabel.CENTER);
      panel.add(msglabel);

      msglabel.addComponentListener(new CustomComponentListener());
      controlPanel.add(panel);
      mainFrame.setVisible(true);
   }

   class CustomComponentListener implements ComponentListener {

      public void componentResized(ComponentEvent e) {
         statusLabel.setText(statusLabel.getText()
         + e.getComponent().getClass().getSimpleName() + " resized. ");
      }

      public void componentMoved(ComponentEvent e) {
         statusLabel.setText(statusLabel.getText()
         + e.getComponent().getClass().getSimpleName() + " moved. ");
      }

      public void componentShown(ComponentEvent e) {
         statusLabel.setText(statusLabel.getText()
         + e.getComponent().getClass().getSimpleName() + " shown. ");
      }

      public void componentHidden(ComponentEvent e) {
         statusLabel.setText(statusLabel.getText()
         + e.getComponent().getClass().getSimpleName() + " hidden. ");
      }
   }
}
```

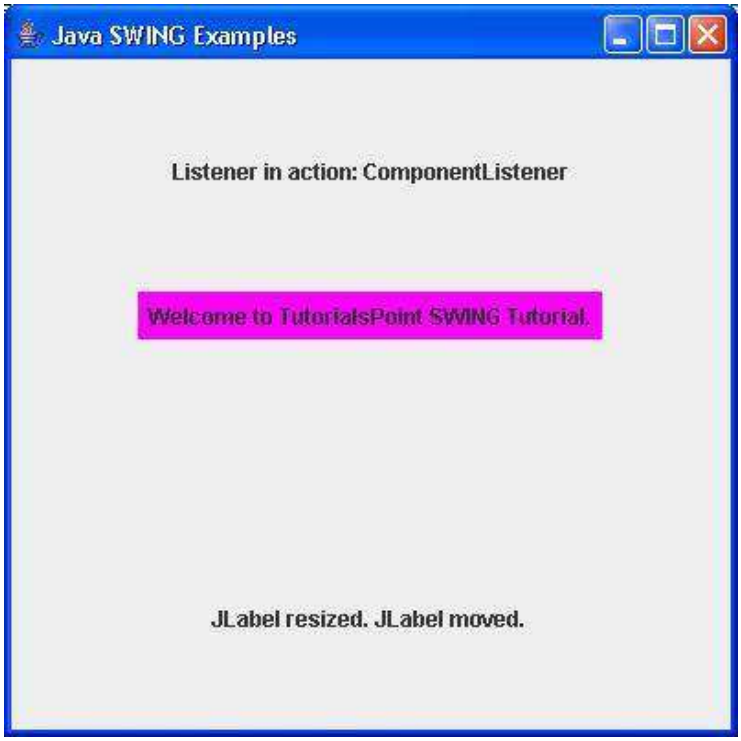
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingListenerDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingListenerDemo
```

Verify the following output.



ItemListener Interface

The class which processes the ItemEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using the **addItemListener()** method. When the action event occurs, that object's itemStateChanged method is invoked.

Interface Declaration

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

```
public interface ItemListener

   extends EventListener
```

Interface Methods

Sr.No.	Method & Description
1	**void itemStateChanged(ItemEvent e)** Invoked when an item has been selected or deselected by the user.

Methods Inherited

This interface inherits methods from the following interface:

- java.awt.EventListener

ItemListener Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingListenerDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingListenerDemo  swingListenerDemo = new SwingListenerDemo();
      swingListenerDemo.showItemListenerDemo();
   }

   private void prepareGUI(){
```

```
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
             System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showItemListenerDemo(){
      headerLabel.setText("Listener in action: ItemListener");
      JCheckBox chkApple = new JCheckBox("Apple");
      JCheckBox chkMango = new JCheckBox("Mango");
      JCheckBox chkPeer = new JCheckBox("Peer");

      chkApple.addItemListener(new CustomItemListener());
      chkMango.addItemListener(new CustomItemListener());
      chkPeer.addItemListener(new CustomItemListener());

      controlPanel.add(chkApple);
      controlPanel.add(chkMango);
      controlPanel.add(chkPeer);
      mainFrame.setVisible(true);
   }
```


```
   class CustomItemListener implements ItemListener {
      public void itemStateChanged(ItemEvent e) {
         statusLabel.setText(((JCheckBox)e.getItem()).getText()
         +" Checkbox: "
         + (e.getStateChange()==1?"checked":"unchecked"));
      }
   }
}
```

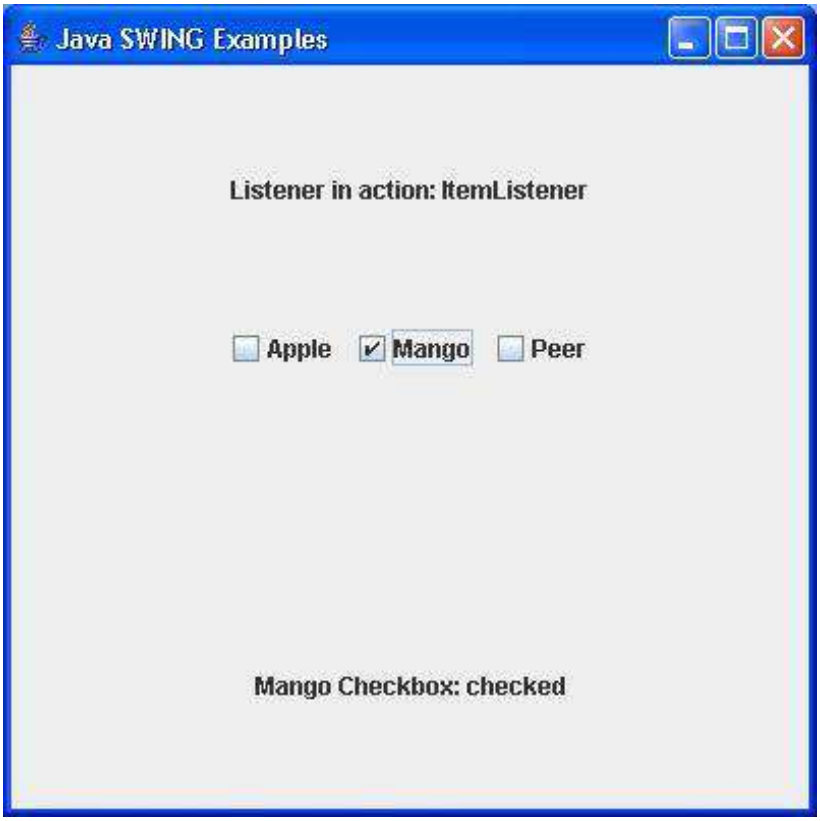
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingListenerDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingListenerDemo
```

Verify the following output.



KeyListener Interface

The class which processes the KeyEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using the **addKeyListener()** method.

Interface Declaration

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

```
public interface KeyListener

   extends EventListener
```

Interface Methods

Sr.No.	Method & Description
1	**void keyPressed(KeyEvent e)** Invoked when a key has been pressed.
2	**void keyReleased(KeyEvent e)** Invoked when a key has been released.
3	**void keyTyped(KeyEvent e)** Invoked when a key has been typed.

Methods Inherited

This interface inherits methods from the following interface:

- java.awt.EventListener

KeyListener Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingListenerDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingListenerDemo(){
      prepareGUI();
   }
```

```
   public static void main(String[] args){
      SwingListenerDemo  swingListenerDemo = new SwingListenerDemo();
      swingListenerDemo.showKeyListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
             System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private JTextField textField;
   private void showKeyListenerDemo(){
      headerLabel.setText("Listener in action: KeyListener");
      textField  = new JTextField(10);

      textField.addKeyListener(new CustomKeyListener());
      JButton okButton = new JButton("OK");
      okButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
```

```
            statusLabel.setText("Entered text: "
               + textField.getText());
         }
      });

      controlPanel.add(textField);
      controlPanel.add(okButton);
      mainFrame.setVisible(true);
   }

   class CustomKeyListener implements KeyListener{
      public void keyTyped(KeyEvent e) {
      }

      public void keyPressed(KeyEvent e) {
         if(e.getKeyCode() == KeyEvent.VK_ENTER){
            statusLabel.setText("Entered text: "
               + textField.getText());
         }
      }

      public void keyReleased(KeyEvent e) {
      }
   }
}
```

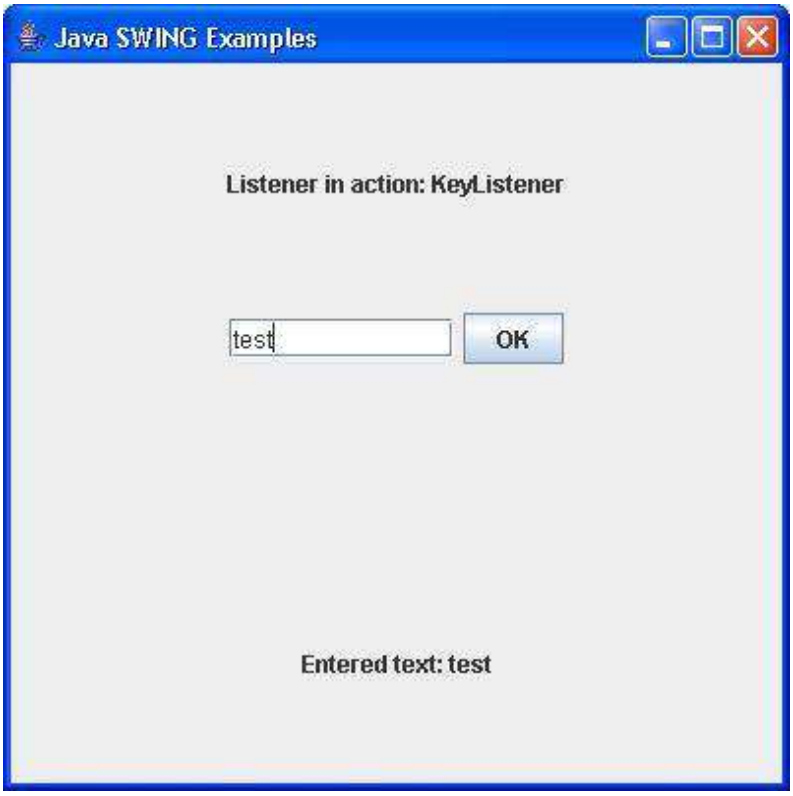
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingListenerDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingListenerDemo
```

Verify the following output.



MouseListener Interface

The class which processes the MouseEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using the **addMouseListener()** method.

Interface Declaration

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

```
public interface MouseListener
   extends EventListener
```

Interface Methods

Sr.No.	Method & Description
1	**void mouseClicked(MouseEvent e)** Invoked when the mouse button has been clicked (pressed and released) on a component.
2	**void mouseEntered(MouseEvent e)**

	Invoked when the mouse enters a component.
3	**void mouseExited(MouseEvent e)** Invoked when the mouse exits a component.
4	**void mousePressed(MouseEvent e)** Invoked when a mouse button has been pressed on a component.
5	**void mouseReleased(MouseEvent e)** Invoked when a mouse button has been released on a component.

Methods Inherited

This interface inherits methods from the following interface:

- java.awt.EventListener

MouseListener Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingListenerDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingListenerDemo(){
      prepareGUI();
   }

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

```
      SwingListenerDemo  swingListenerDemo = new SwingListenerDemo();
      swingListenerDemo.showMouseListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
             System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showMouseListenerDemo(){
      headerLabel.setText("Listener in action: MouseListener");

      JPanel panel = new JPanel();
      panel.setBackground(Color.magenta);
      panel.setLayout(new FlowLayout());
      panel.addMouseListener(new CustomMouseListener());

      JLabel msglabel =
      new JLabel("Welcome to TutorialsPoint SWING Tutorial."
```

```
      ,JLabel.CENTER);
      panel.add(msglabel);

      msglabel.addMouseListener(new CustomMouseListener());
      panel.add(msglabel);

      controlPanel.add(panel);
      mainFrame.setVisible(true);
   }

   class CustomMouseListener implements MouseListener{
      public void mouseClicked(MouseEvent e) {
         statusLabel.setText("Mouse Clicked: ("+e.getX()+", "+e.getY() +")");
      }

      public void mousePressed(MouseEvent e) {
      }

      public void mouseReleased(MouseEvent e) {
      }

      public void mouseEntered(MouseEvent e) {
      }

      public void mouseExited(MouseEvent e) {
      }
   }
}
```

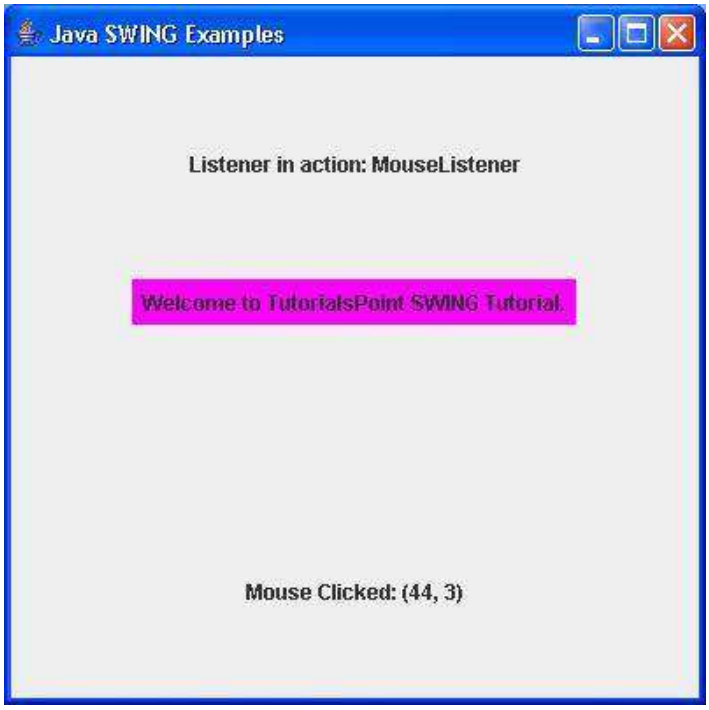
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingListenerDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingListenerDemo
```

Verify the following output.



WindowListener Interface

The class which processes the WindowEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using the **addWindowListener()** method.

Interface Declaration

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

```
public interface WindowListener
   extends EventListener
```

Interface Methods

Sr.No.	Method & Description
1	**void windowActivated(WindowEvent e)** Invoked when the Window is set to be the active Window.
2	**void windowClosed(WindowEvent e)** Invoked when a window has been closed as a result of calling dispose on the Window.

3	**void windowClosing(WindowEvent e)** Invoked when the user attempts to close the Window from the Window's system menu.
4	**void windowDeactivated(WindowEvent e)** Invoked when a Window is no longer the active Window.
5	**void windowDeiconified(WindowEvent e)** Invoked when a Window is changed from a minimized to a normal state.
6	**void windowIconified(WindowEvent e)** Invoked when a Window is changed from a normal to a minimized state.
7	**void windowOpened(WindowEvent e)** Invoked the first time a Window is made visible.

Methods Inherited

This interface inherits methods from the following interface:

- java.awt.EventListener

WindowListener Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingListenerDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;
```

```
public SwingListenerDemo(){
   prepareGUI();
}

public static void main(String[] args){
   SwingListenerDemo  swingListenerDemo = new SwingListenerDemo();
   swingListenerDemo.showWindowListenerDemo();
}

private void prepareGUI(){
   mainFrame = new JFrame("Java SWING Examples");
   mainFrame.setSize(400,400);
   mainFrame.setLayout(new GridLayout(3, 1));

   headerLabel = new JLabel("",JLabel.CENTER );
   statusLabel = new JLabel("",JLabel.CENTER);

   statusLabel.setSize(350,100);
   mainFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent windowEvent){
          System.exit(0);
      }
   });
   controlPanel = new JPanel();
   controlPanel.setLayout(new FlowLayout());

   mainFrame.add(headerLabel);
   mainFrame.add(controlPanel);
   mainFrame.add(statusLabel);
   mainFrame.setVisible(true);
}

private void showWindowListenerDemo(){
   headerLabel.setText("Listener in action: WindowListener");

   JButton okButton = new JButton("OK");
```

```
      aboutFrame = new JFrame();
      aboutFrame.setSize(300,200);;
      aboutFrame.setTitle("WindowListener Demo");
      aboutFrame.addWindowListener(new CustomWindowListener());

      JPanel panel = new JPanel();
      panel.setBackground(Color.white);
      JLabel msglabel
      = new JLabel("Welcome to TutorialsPoint SWING Tutorial."
      ,JLabel.CENTER);
      panel.add(msglabel);
      aboutFrame.add(panel);
      aboutFrame.setVisible(true);
   }
   class CustomWindowListener implements WindowListener {
      public void windowOpened(WindowEvent e) {
      }

      public void windowClosing(WindowEvent e) {
         aboutFrame.dispose();
      }

      public void windowClosed(WindowEvent e) {
      }

      public void windowIconified(WindowEvent e) {
      }

      public void windowDeiconified(WindowEvent e) {
      }

      public void windowActivated(WindowEvent e) {
      }

      public void windowDeactivated(WindowEvent e) {
      }
   }
```

```
}
```

Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingListenerDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingListenerDemo
```

Verify the following output.

AdjustmentListener Interface

Introduction

The interface **AdjustmentListener** is used for receiving adjustment events. The class that processes adjustment events needs to implement this interface.

Class Declaration

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

```
public interface AdjustmentListener
```

```
extends EventListener
```

Interface Methods

Sr.No.	Method & Description
1	**void adjustmentValueChanged(AdjustmentEvent e)** Invoked when the value of the adjustable has changed.

Methods Inherited

This class inherits methods from the following interface:

- java.awt.event.EventListener

AdjustmentListener Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingListenerDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingListenerDemo  swingListenerDemo = new SwingListenerDemo();
```

```
      swingListenerDemo.showAdjustmentListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
             System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showAdjustmentListenerDemo(){
      headerLabel.setText("Listener in action: AdjustmentListener");

      JPanel panel = new JPanel();
      JScrollBar scrollbar = new JScrollBar();
      scrollbar.addAdjustmentListener(new CustomAdjustmentListener());

      panel.add(scrollbar);
      controlPanel.add(panel);
      mainFrame.setVisible(true);
   }
```

```
      class CustomAdjustmentListener implements AdjustmentListener {
         public void adjustmentValueChanged(AdjustmentEvent e) {
            statusLabel.setText("Adjustment value: "
               +Integer.toString(e.getValue()));
         }
      }
}
```

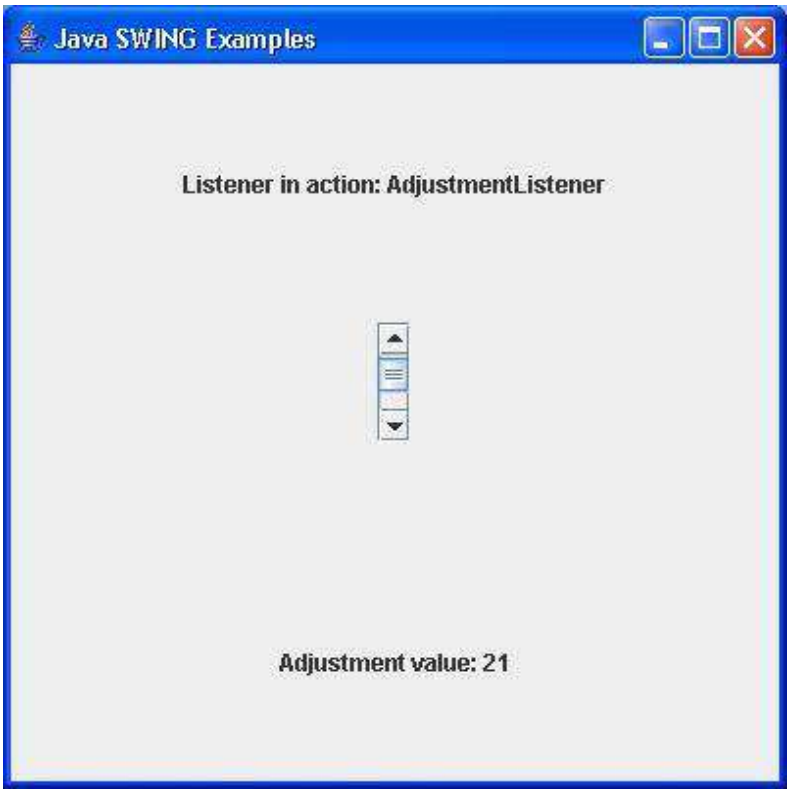
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingListenerDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingListenerDemo
```

Verify the following output.



ContainerListener Interface

Introduction

The interface **ContainerListener** is used for receiving container events. The class that processes container events needs to implements this interface.

Class Declaration

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

```
public interface ContainerListener
extends EventListener
```

Interface Methods

Sr.No.	Method & Description
1	**void componentAdded(ContainerEvent e)** Invoked when a component has been added to the container.
2	**void componentRemoved(ContainerEvent e)** Invoked when a component has been removed from the container.

Methods Inherited

This class inherits methods from the following interface:

- java.awt.event.EventListener

ContainerListener Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingListenerDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
```

```
   private JPanel controlPanel;

   public SwingListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingListenerDemo  swingListenerDemo = new SwingListenerDemo();
      swingListenerDemo.showContainerListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
             System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showContainerListenerDemo(){
      headerLabel.setText("Listener in action: ContainerListener");
```

```
      JPanel panel = new JPanel();
      panel.setBackground(Color.magenta);
      panel.addContainerListener(new CustomContainerListener());

      JLabel msglabel
      = new JLabel("Welcome to TutorialsPoint SWING Tutorial."
      ,JLabel.CENTER);
      panel.add(msglabel);

      controlPanel.add(panel);
      mainFrame.setVisible(true);
   }

   class CustomContainerListener implements ContainerListener {
      public void componentAdded(ContainerEvent e) {
         statusLabel.setText(statusLabel.getText()
         + e.getComponent().getClass().getSimpleName() + " added. ");
      }

      public void componentRemoved(ContainerEvent e) {
         statusLabel.setText(statusLabel.getText()
         + e.getComponent().getClass().getSimpleName() + " removed. ");
      }
   }

}
```

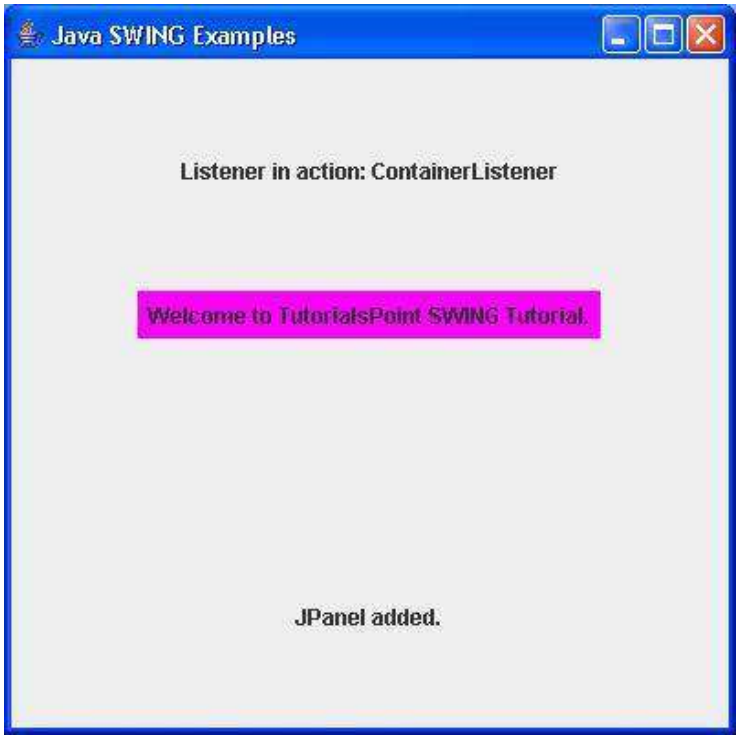
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingListenerDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingListenerDemo
```

Verify the following output.



MouseMotionListener Interface

Introduction

The interface **MouseMotionListener** is used for receiving mouse motion events on a component. The class that processes mouse motion events needs to implements this interface.

Class Declaration

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

```
public interface MouseMotionListener

extends EventListener
```

Interface Methods

Sr.No.	Method & Description
1	**void mouseDragged(MouseEvent e)** Invoked when a mouse button is pressed on a component and then dragged.
2	**void mouseMoved(MouseEvent e)** Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.

Methods Inherited

This class inherits methods from the following interface:

- java.awt.event.EventListener

MouseMotionListener Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingListenerDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingListenerDemo(){
      prepareGUI();
   }
```

```
   public static void main(String[] args){
      SwingListenerDemo  swingListenerDemo = new SwingListenerDemo();
      swingListenerDemo.showMouseMotionListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
              System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showMouseMotionListenerDemo(){
      headerLabel.setText("Listener in action: MouseMotionListener");
```

```
      JPanel panel = new JPanel();
      panel.setBackground(Color.magenta);
      panel.setLayout(new FlowLayout());
      panel.addMouseMotionListener(new CustomMouseMotionListener());

      JLabel msglabel
      = new JLabel("Welcome to TutorialsPoint SWING Tutorial."
      ,JLabel.CENTER);
      panel.add(msglabel);

      controlPanel.add(panel);
      mainFrame.setVisible(true);
   }

   class CustomMouseMotionListener implements MouseMotionListener {
      public void mouseDragged(MouseEvent e) {
         statusLabel.setText("Mouse Dragged: ("+e.getX()+", "+e.getY() +")");
      }

      public void mouseMoved(MouseEvent e) {
         statusLabel.setText("Mouse Moved: ("+e.getX()+", "+e.getY() +")");
      }
   }
}
```

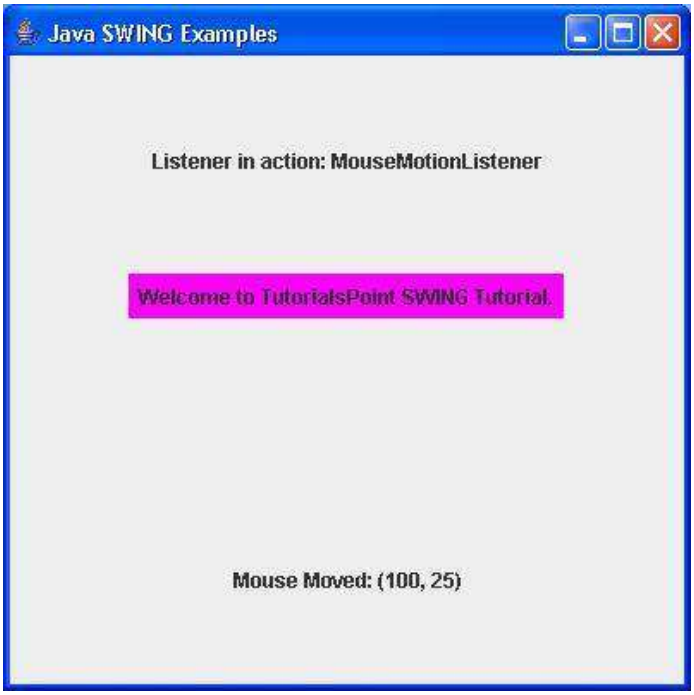
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingListenerDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingListenerDemo
```

Verify the following output.



FocusListener Interface

Introduction

The interface **FocusListener** is used for receiving keyboard focus events. The class that processes focus events needs to implement this interface.

Class Declaration

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

```
public interface FocusListener
extends EventListener
```

Interface Methods

Sr.No.	Method & Description
1	**void focusGained(FocusEvent e)** Invoked when a component gains the keyboard focus.
2	**void focusLost(FocusEvent e)** Invoked when a component loses the keyboard focus.

Methods Inherited

This class inherits methods from the following interface:

- java.awt.event.EventListener

FocusListener Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingListenerDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingListenerDemo  swingListenerDemo = new SwingListenerDemo();
      swingListenerDemo.showFocusListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);
```

```
      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
             System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }
   private void showFocusListenerDemo(){

      headerLabel.setText("Listener in action: FocusListener");
      JButton okButton = new JButton("OK");
      JButton cancelButton = new JButton("Cancel");

      okButton.addFocusListener(new CustomFocusListener());
      cancelButton.addFocusListener(new CustomFocusListener());

      controlPanel.add(okButton);
      controlPanel.add(cancelButton);
      mainFrame.setVisible(true);

   }

   class CustomFocusListener implements FocusListener{
      public void focusGained(FocusEvent e) {
         statusLabel.setText(statusLabel.getText()
         + e.getComponent().getClass().getSimpleName() + " gained focus. ");
      }

      public void focusLost(FocusEvent e) {
         statusLabel.setText(statusLabel.getText()
```

```
         + e.getComponent().getClass().getSimpleName() + " lost focus. ");
      }
   }
}
```

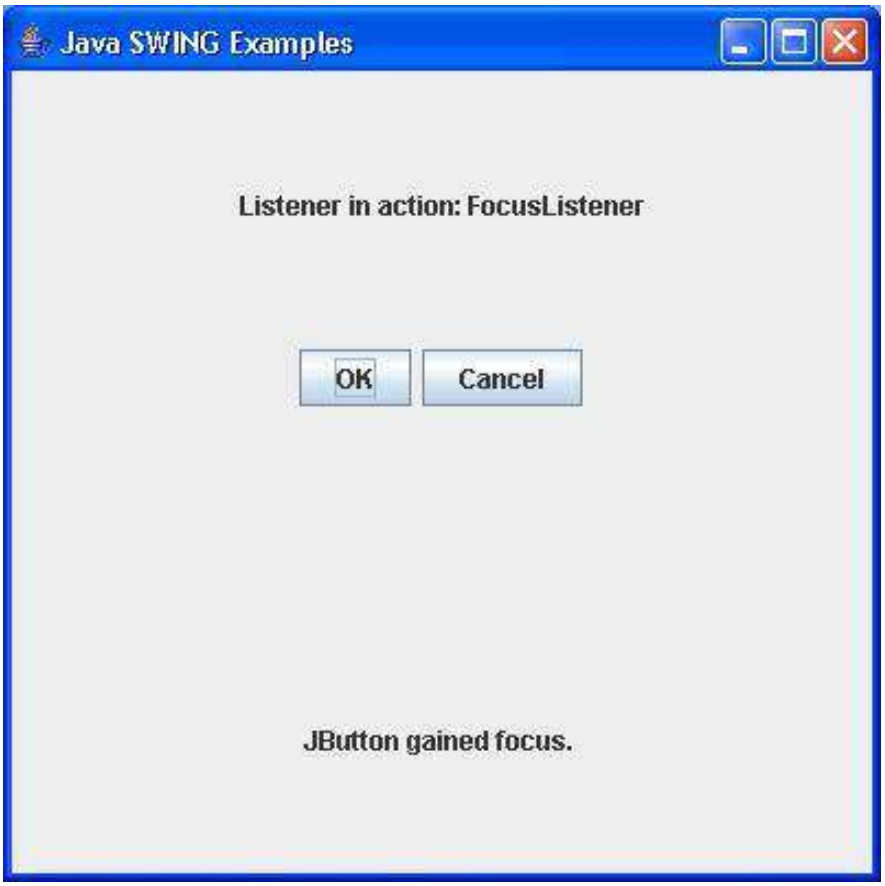
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingListenerDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingListenerDemo
```

Verify the following output.



7. Swing – Event Adapters

Adapters are abstract classes for receiving various events. The methods in these classes are empty. These classes exist as convenience for creating listener objects.

SWING Adapters

Following is the list of commonly used adapters while listening GUI events in SWING.

Sr. No.	Adapter & Description
1	**FocusAdapter** An abstract adapter class for receiving focus events.
2	**KeyAdapter** An abstract adapter class for receiving key events.
3	**MouseAdapter** An abstract adapter class for receiving mouse events.
4	**MouseMotionAdapter** An abstract adapter class for receiving mouse motion events.
5	**WindowAdapter** An abstract adapter class for receiving window events.

FocusAdapter Class

Introduction

The class **FocusAdapter** is an abstract (adapter) class for receiving keyboard focus events. All methods of this class are empty. This class is convenience class for creating listener objects.

Class Declaration

Following is the declaration for **java.awt.event.FocusAdapter** class:

```
public abstract class FocusAdapter
   extends Object
      implements FocusListener
```

Class Constructors

Sr.No.	Constructor & Description
1	**FocusAdapter()**

Class Methods

Sr.No.	Method & Description
1	**void focusGained(FocusEvent e)** Invoked when a component gains the keyboard focus.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

FocusAdapter Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingAdapterDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingAdapterDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingAdapterDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingAdapterDemo  swingAdapterDemo = new SwingAdapterDemo();
      swingAdapterDemo.showFocusAdapterDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);
```

```
      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
             System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showFocusAdapterDemo(){

      headerLabel.setText("Listener in action: FocusAdapter");

      JButton okButton = new JButton("OK");
      JButton cancelButton = new JButton("Cancel");
      okButton.addFocusListener(new FocusAdapter(){
         public void focusGained(FocusEvent e) {
            statusLabel.setText(statusLabel.getText()
            + e.getComponent().getClass().getSimpleName()
            + " gained focus. ");
         }
      });
```

```
      cancelButton.addFocusListener(new FocusAdapter(){
         public void focusLost(FocusEvent e) {
            statusLabel.setText(statusLabel.getText()
            + e.getComponent().getClass().getSimpleName()
            + " lost focus. ");
         }
      });

      controlPanel.add(okButton);
      controlPanel.add(cancelButton);
      mainFrame.setVisible(true);
   }
}
```

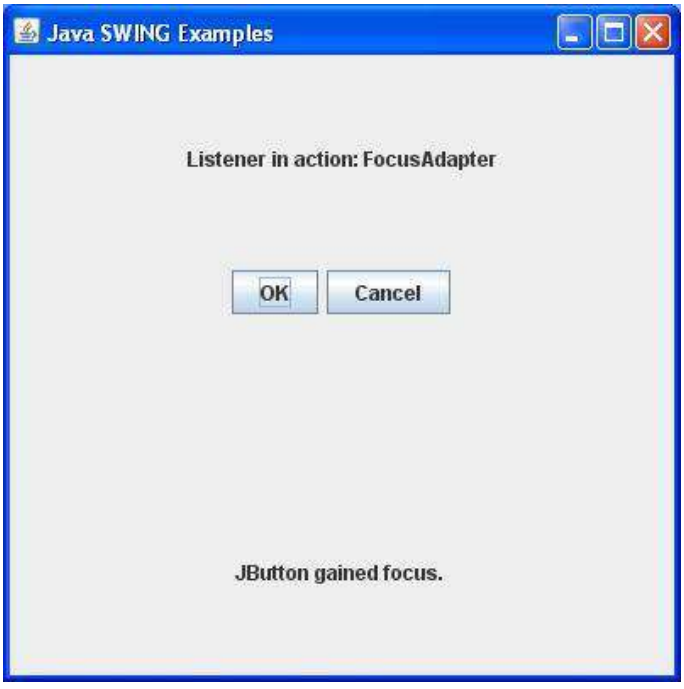
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingAdapterDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingAdapterDemo
```

Verify the following output.



KeyAdapter Class

Introduction

The class **KeyAdapter** is an abstract (adapter) class for receiving keyboard events. All methods of this class are empty. This class is convenience class for creating listener objects.

Class Declaration

Following is the declaration for **java.awt.event.KeyAdapter** class:

```
public abstract class KeyAdapter
   extends Object
      implements KeyListener
```

Class Constructors

Sr.No.	Constructor & Description
1	**KeyAdapter()**

Class Methods

Sr.No.	Method & Description
1	**void keyPressed(KeyEvent e)** Invoked when a key has been pressed.
2	**void keyReleased(KeyEvent e)** Invoked when a key has been released.
3	**void keyTyped(KeyEvent e)** Invoked when a key has been typed.

Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

KeyAdapter Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingAdapterDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingAdapterDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingAdapterDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingAdapterDemo  swingAdapterDemo = new SwingAdapterDemo();
      swingAdapterDemo.showKeyAdapterDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
```

```
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
             System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showKeyAdapterDemo(){
      headerLabel.setText("Listener in action: KeyAdapter");

      final JTextField textField = new JTextField(10);

      textField.addKeyListener(new KeyAdapter() {
         public void keyPressed(KeyEvent e) {
            if(e.getKeyCode() == KeyEvent.VK_ENTER){
               statusLabel.setText("Entered text: "
               + textField.getText());
            }
         }
      });
      JButton okButton = new JButton("OK");
      okButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            statusLabel.setText("Entered text: "
            + textField.getText());
         }
      });

      controlPanel.add(textField);
      controlPanel.add(okButton);
```

```
      mainFrame.setVisible(true);
   }
}
```

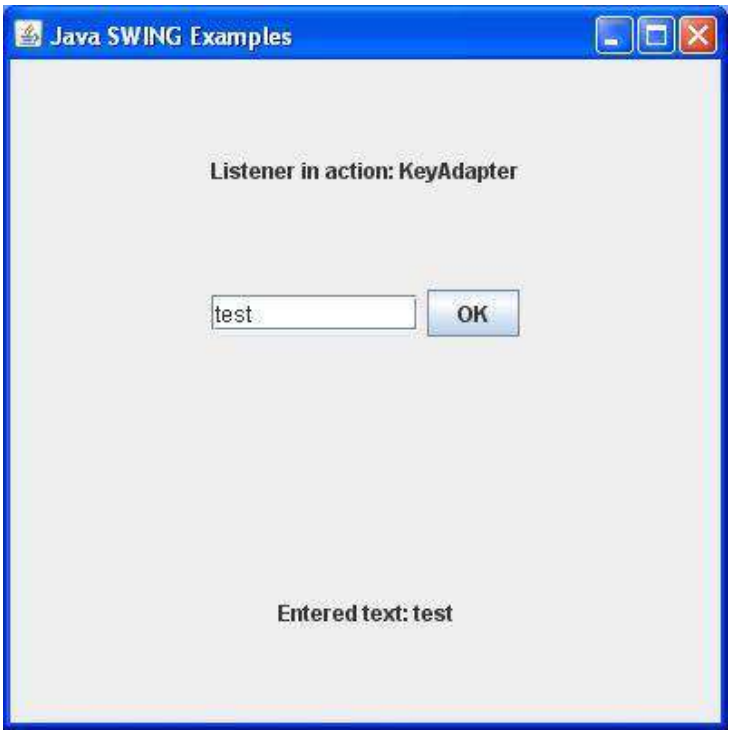
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingAdapterDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingAdapterDemo
```

Verify the following output.



MouseAdapter Class

Introduction

The class **MouseAdapter** is an abstract (adapter) class for receiving mouse events. All methods of this class are empty. This class is convenience class for creating listener objects.

Class Declaration

Following is the declaration for **java.awt.event.MouseAdapter** class:

```
public abstract class MouseAdapter
   extends Object
      implements MouseListener, MouseWheelListener, MouseMotionListener
```

Class Constructors

Sr.No.	Constructor & Description
1	**MouseAdapter()**

Class Methods

Sr.No.	Method & Description
1	**void mouseClicked(MouseEvent e)** Invoked when the mouse button has been clicked (pressed and released) on a component.
2	**void mouseDragged(MouseEvent e)** Invoked when a mouse button is pressed on a component and then dragged.
3	**void mouseEntered(MouseEvent e)** Invoked when the mouse enters a component.
4	**void mouseExited(MouseEvent e)** Invoked when the mouse exits a component.
5	**void mouseMoved(MouseEvent e)** Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.
6	**void mousePressed(MouseEvent e)** Invoked when a mouse button has been pressed on a component.
7	**void mouseReleased(MouseEvent e)**

	Invoked when a mouse button has been released on a component.
8	**void mouseWheelMoved(MouseWheelEvent e)** Invoked when the mouse wheel is rotated.

Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

MouseAdapter Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingAdapterDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingAdapterDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingAdapterDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtAdapterDemo  awtAdapterDemo = new AwtAdapterDemo();
      swingAdapterDemo.showMouseAdapterDemo();
   }
```

```
private void prepareGUI(){
   mainFrame = new JFrame("Java SWING Examples");
   mainFrame.setSize(400,400);
   mainFrame.setLayout(new GridLayout(3, 1));

   headerLabel = new JLabel("",JLabel.CENTER );
   statusLabel = new JLabel("",JLabel.CENTER);

   statusLabel.setSize(350,100);
   mainFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent windowEvent){
          System.exit(0);
      }
   });
   controlPanel = new JPanel();
   controlPanel.setLayout(new FlowLayout());

   mainFrame.add(headerLabel);
   mainFrame.add(controlPanel);
   mainFrame.add(statusLabel);
   mainFrame.setVisible(true);
}

private void showMouseAdapterDemo(){
   headerLabel.setText("Listener in action: MouseAdapter");

   JPanel panel = new JPanel();
   panel.setBackground(Color.magenta);
   panel.setLayout(new FlowLayout());
   panel.addMouseListener(new MouseAdapter(){
      public void mouseClicked(MouseEvent e) {
         statusLabel.setText("Mouse Clicked: ("
         +e.getX()+", "+e.getY() +")");
      }
   });
```

```
      JLabel msglabel
      = new JLabel("Welcome to TutorialsPoint SWING Tutorial."
      ,JLabel.CENTER);

      msglabel.addMouseListener(new MouseAdapter(){
         public void mouseClicked(MouseEvent e) {
            statusLabel.setText("Mouse Clicked: ("
            +e.getX()+", "+e.getY() +")");
         }
      });
      panel.add(msglabel);
      controlPanel.add(panel);
      mainFrame.setVisible(true);
   }
}
```

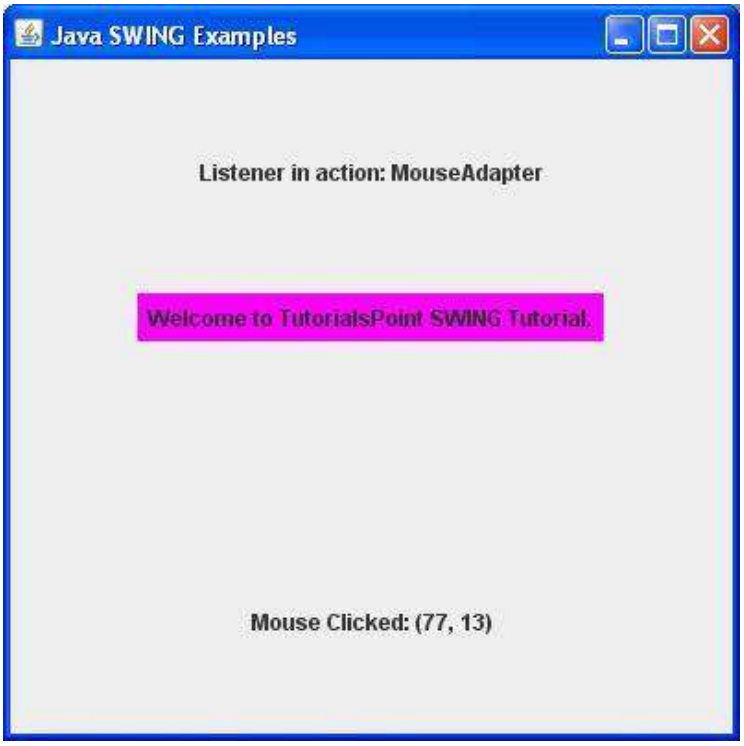
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingAdapterDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingAdapterDemo
```

Verify the following output.



MouseMotionAdapter Class

Introduction

The class **MouseMotionAdapter** is an abstract (adapter) class for receiving mouse motion events. All methods of this class are empty. This class is convenience class for creating listener objects.

Class Declaration

Following is the declaration for **java.awt.event.MouseMotionAdapter** class:

```
public abstract class MouseMotionAdapter
   extends Object
      implements MouseMotionListener
```

Class Constructors

Sr.No.	Constructor & Description
1	**MouseMotionAdapter()**

Class Methods

Sr.No.	Method & Description
1	**void mouseDragged(MouseEvent e)** Invoked when a mouse button is pressed on a component and then dragged.
2	**void mouseMoved(MouseEvent e)** Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

MouseMotionAdapter Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingAdapterDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingAdapterDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
```

```
   private JPanel controlPanel;

   public SwingAdapterDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingAdapterDemo  swingAdapterDemo = new SwingAdapterDemo();
      swingAdapterDemo.showMouseMotionAdapterDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
             System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showMouseMotionAdapterDemo(){
      headerLabel.setText("Listener in action: MouseMotionAdapter");
```

```
      JPanel panel = new JPanel();
      panel.setBackground(Color.magenta);
      panel.setLayout(new FlowLayout());
      panel.addMouseMotionListener(new MouseMotionAdapter(){
         public void mouseMoved(MouseEvent e) {
            statusLabel.setText("Mouse Moved: ("+e.getX()+", "+e.getY() +")");
         }
      });

      JLabel msglabel
      = new JLabel("Welcome to TutorialsPoint SWING Tutorial."
      ,JLabel.CENTER);
      panel.add(msglabel);

      controlPanel.add(panel);

      mainFrame.setVisible(true);
   }
}
```

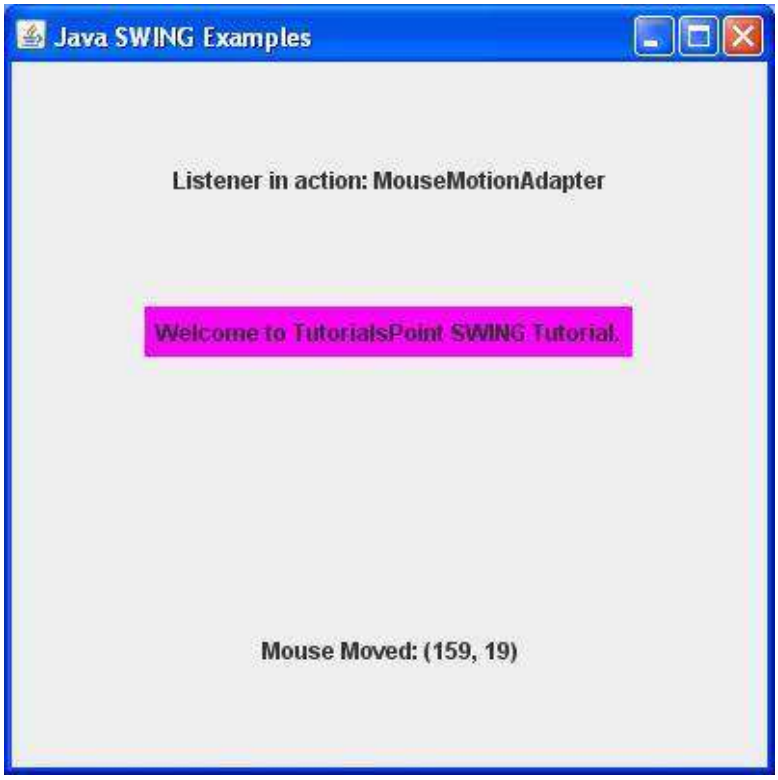
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingAdapterDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingAdapterDemo
```

Verify the following output.



WindowAdapter Class

Introduction

The class **WindowAdapter** is an abstract (adapter) class for receiving window events. All methods of this class are empty. This class is convenience class for creating listener objects.

Class Declaration

Following is the declaration for **java.awt.event.WindowAdapter** class:

```
public abstract class WindowAdapter
   extends Object
      implements WindowListener, WindowStateListener, WindowFocusListener
```

Class Constructors

Sr.No.	Constructor & Description
1	**WindowAdapter()**

Class Methods

Sr.No.	Method & Description
1	**void windowActivated(WindowEvent e)** Invoked when a Window is activated.
2	**void windowClosed(WindowEvent e)** Invoked when a Window has been closed.
3	**void windowClosing(WindowEvent e)** Invoked when a Window is in the process of being closed.
4	**void windowDeactivated(WindowEvent e)** Invoked when a Window is de-activated.
5	**void windowDeiconified(WindowEvent e)** Invoked when a Window is de-iconified.
6	**void windowGainedFocus(WindowEvent e)** Invoked when the Window is set to be the focused Window, which means that the Window, or one of its subcomponents, will receive keyboard events.
7	**void windowIconified(WindowEvent e)** Invoked when a Window is iconified.
8	**void windowLostFocus(WindowEvent e)** Invoked when the Window is no longer the focused Window, which means that keyboard events will no longer be delivered to the Window or any of its subcomponents.
9	**void windowOpened(WindowEvent e)** Invoked when a Window has been opened.
10	**void windowStateChanged(WindowEvent e)** Invoked when a Window state is changed.

Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

WindowAdapter Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingAdapterDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingAdapterDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingAdapterDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingAdapterDemo  swingAdapterDemo = new SwingAdapterDemo();
      swingAdapterDemo.showWindowAdapterDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
```

```
            System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showWindowAdapterDemo(){
      headerLabel.setText("Listener in action: WindowAdapter");

      JButton okButton = new JButton("OK");

      final JFrame aboutFrame = new JFrame();
      aboutFrame.setSize(300,200);;
      aboutFrame.setTitle("WindowAdapter Demo");
      aboutFrame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent windowEvent){
               aboutFrame.dispose();
         }
      });
      JLabel msglabel
      = new JLabel("Welcome to TutorialsPoint SWING Tutorial."
      ,JLabel.CENTER);
       aboutFrame.add(msgLabel);
      aboutFrame.setVisible(true);
   }
}
```

Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingAdapterDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingAdapterDemo
```

Verify the following output.

8. Swing – Layouts

Layout refers to the arrangement of components within the container. In another way, it could be said that layout is placing the components at a particular position within the container. The task of laying out the controls is done automatically by the Layout Manager.

Layout Manager

The layout manager automatically positions all the components within the container. Even if you do not use the layout manager, the components are still positioned by the default layout manager. It is possible to lay out the controls by hand, however, it becomes very difficult because of the following two reasons.

- It is very tedious to handle a large number of controls within the container.
- Usually, the width and height information of a component is not given when we need to arrange them.

Java provides various layout managers to position the controls. Properties like size, shape, and arrangement varies from one layout manager to the other. When the size of the applet or the application window changes, the size, shape, and arrangement of the components also changes in response, i.e. the layout managers adapt to the dimensions of the appletviewer or the application window.

The layout manager is associated with every Container object. Each layout manager is an object of the class that implements the LayoutManager interface.

Following are the interfaces defining the functionalities of Layout Managers.

Sr. No.	Interface & Description
1	**LayoutManager** The LayoutManager interface declares those methods which need to be implemented by the class, whose object will act as a layout manager.
2	**LayoutManager2** The LayoutManager2 is the sub-interface of the LayoutManager. This interface is for those classes that know how to layout containers based on lay out constraint object.

LayoutManager Interface

Introduction

The interface **LayoutManager** is used to define the interface for classes that know how to lay out Containers.

Class Declaration

Following is the declaration for **java.awt.LayoutManager** interface:

```
public interface LayoutManager
```

Interface Methods

Sr.No.	Method & Description
1	**void addLayoutComponent(String name, Component comp)** If the layout manager uses a per-component string, adds the component comp to the layout, associating it with the string specified by name.
2	**void layoutContainer(Container parent)** Lays out the specified container.
3	**Dimension minimumLayoutSize(Container parent)** Calculates the minimum size dimensions for the specified container, given the components it contains.
4	**Dimension preferredLayoutSize(Container parent)** Calculates the preferred size dimensions for the specified container, given the components it contains.
5	**void removeLayoutComponent(Component comp)** Removes the specified component from the layout.

LayoutManager2 Interface

Introduction

The interface **LayoutManager** is used to define the interface for classes that know how to lay out containers based on a layout constraints object.

Class Declaration

Following is the declaration for **java.awt.LayoutManager2** interface:

```
public interface LayoutManger2
   extends LayoutManager
```

Interface Methods

Sr.No.	Method & Description
1	**void addLayoutComponent(Component comp, Object constraints)** Adds the specified component to the layout, using the specified constraint object.
2	**float getLayoutAlignmentX(Container target)** Returns the alignment along the **x** axis.
3	**float getLayoutAlignmentY(Container target)** Returns the alignment along the **y** axis.
4	**void invalidateLayout(Container target)** Invalidates the layout, indicating that if the layout manager has cached information it should be discarded.
5	**Dimension maximumLayoutSize(Container target)** Calculates the maximum size dimensions for the specified container, given the components it contains.

AWT Layout Manager Classes

Following is the list of commonly used controls while designing GUI using AWT.

Sr. No.	LayoutManager & Description
1	**BorderLayout** The borderlayout arranges the components to fit in the five regions: east, west, north, south, and center.
2	**CardLayout** The CardLayout object treats each component in the container as a card. Only one card is visible at a time.
3	**FlowLayout** The FlowLayout is the default layout. It lays out the components in a directional flow.
4	**GridLayout** The GridLayout manages the components in the form of a rectangular grid.
5	**GridBagLayout** This is the most flexible layout manager class. The object of GridBagLayout aligns the component vertically, horizontally, or along their baseline without requiring the components of the same size.
6	**GroupLayout** The GroupLayout hierarchically groups the components in order to position them in a Container.
7	**SpringLayout** A SpringLayout positions the children of its associated container according to a set of constraints.

BorderLayout Class

Introduction

The class **BorderLayout** arranges the components to fit in the five regions: east, west, north, south, and center. Each region can contain only one component and each component in each region is identified by the corresponding constant NORTH, SOUTH, EAST, WEST, and CENTER.

Class Declaration

Following is the declaration for **java.awt.BorderLayout** class:

```
public class BorderLayout
   extends Object
      implements LayoutManager2, Serializable
```

Field

Following are the fields for **java.awt.BorderLayout** class:

- **static String AFTER_LAST_LINE** - Synonym for PAGE_END.
- **static String AFTER_LINE_ENDS** - Synonym for LINE_END.
- **static String BEFORE_FIRST_LINE** - Synonym for PAGE_START.
- **static String BEFORE_LINE_BEGINS** - Synonym for LINE_START.
- **static String CENTER** - The center layout constraint (middle of the container).
- **static String EAST** - The east layout constraint (right side of the container).
- **static String LINE_END** - The component goes at the end of the line direction for the layout.
- **static String LINE_START** - The component goes at the beginning of the line direction for the layout.
- **static String NORTH** - The north layout constraint (top of the container).
- **static String PAGE_END** - The component comes after the last line of the layout's content.
- **static String PAGE_START** - The component comes before the first line of the layout's content.
- **static String SOUTH** - The south layout constraint (bottom of the container).
- **static String WEST** - The west layout constraint (left side of the container).

Class Constructors

Sr.No.	Constructor & Description
1	**BorderLayout()** Constructs a new border layout with no gaps between the components.
2	**BorderLayout(int hgap, int vgap)** Constructs a border layout with the specified gaps between the components.

Class Methods

Sr.No.	Method & Description
1	**void addLayoutComponent(Component comp, Object constraints)** Adds the specified component to the layout, using the specified constraint object.
2	**void addLayoutComponent(String name, Component comp)** If the layout manager uses a per-component string, adds the component comp to the layout, associating it with the string specified by name.
3	**int getHgap()** Returns the horizontal gap between the components.
4	**float getLayoutAlignmentX(Container parent)** Returns the alignment along the **x** axis.
5	**float getLayoutAlignmentY(Container parent)** Returns the alignment along the **y** axis.
6	**int getVgap()** Returns the vertical gap between the components.
7	**void invalidateLayout(Container target)** Invalidates the layout, indicating that if the layout manager has cached information it should be discarded.

8	**void layoutContainer(Container target)** Lays out the container argument using this border layout.
9	**Dimension maximumLayoutSize(Container target)** Returns the maximum dimensions for this layout given the components in the specified target container.
10	**Dimension minimumLayoutSize(Container target)** Determines the minimum size of the target container using this layout manager.
11	**Dimension preferredLayoutSize(Container target)** Determines the preferred size of the target container using this layout manager, based on the components in the container.
12	**void removeLayoutComponent(Component comp)** Removes the specified component from this border layout.
13	**void setHgap(int hgap)** Sets the horizontal gap between the components.
14	**void setVgap(int vgap)** Sets the vertical gap between the components.
15	**String toString()** Returns a string representation of the state of this border layout.

Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

BorderLayout Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingLayoutDemo.java

```
package com.tutorialspoint.gui;


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

```
import javax.swing.*;

public class SwingLayoutDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;
   private JLabel msglabel;

   public SwingLayoutDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingLayoutDemo swingLayoutDemo = new SwingLayoutDemo();
      swingLayoutDemo.showBorderLayoutDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
             System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
```

```
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showBorderLayoutDemo(){
      headerLabel.setText("Layout in action: BorderLayout");

      JPanel panel = new JPanel();
      panel.setBackground(Color.darkGray);
      panel.setSize(300,300);
      BorderLayout layout = new BorderLayout();
      layout.setHgap(10);
      layout.setVgap(10);
      panel.setLayout(layout);

      panel.add(new JButton("Center"),BorderLayout.CENTER);
      panel.add(new JButton("Line Start"),BorderLayout.LINE_START);
      panel.add(new JButton("Line End"),BorderLayout.LINE_END);
      panel.add(new JButton("East"),BorderLayout.EAST);
      panel.add(new JButton("West"),BorderLayout.WEST);
      panel.add(new JButton("North"),BorderLayout.NORTH);
      panel.add(new JButton("South"),BorderLayout.SOUTH);

      controlPanel.add(panel);
      mainFrame.setVisible(true);
   }
}
```

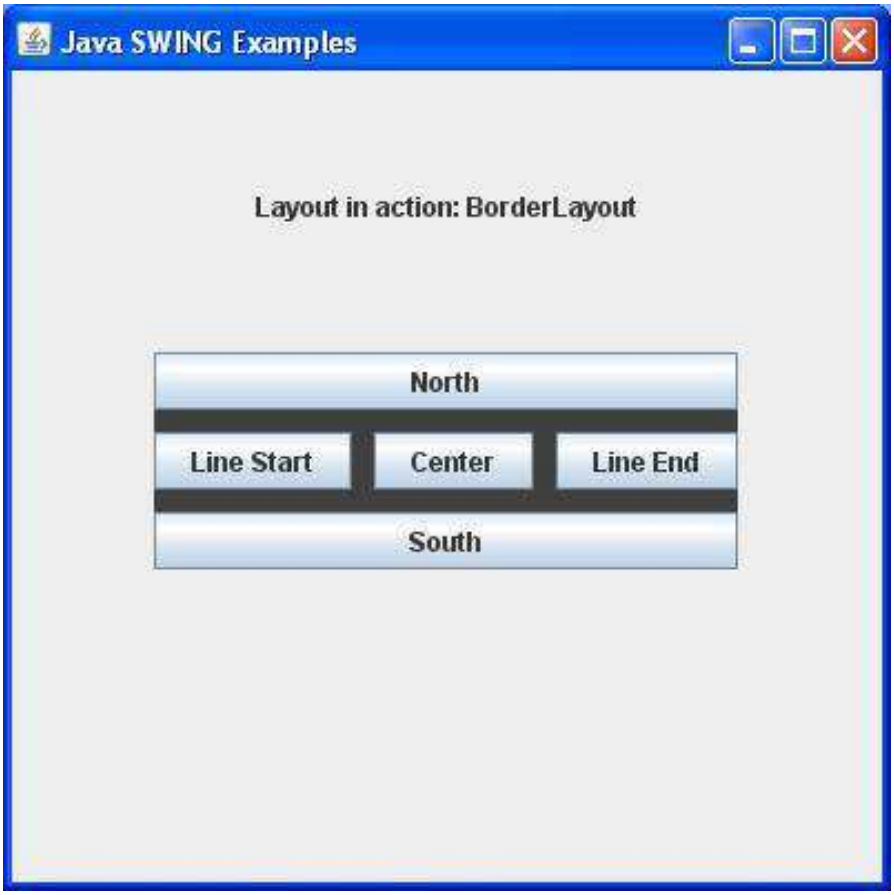
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingLayoutDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingLayoutDemo
```

Verify the following output.



CardLayout Class

Introduction

The class **CardLayout** arranges each component in the container as a card. Only one card is visible at a time, and the container acts as a stack of cards.

Class Declaration

Following is the declaration for **java.awt.CardLayout** class:

```
public class CardLayout
   extends Object
      implements LayoutManager2, Serializable
```

Class Constructors

Sr.No.	Constructor & Description
1	**CardLayout()** Creates a new card layout with the gaps of size zero.

2	**CardLayout(int hgap, int vgap)** Creates a new card layout with the specified horizontal and vertical gaps.

Class Methods

Sr.No.	Method & Description
1	**void addLayoutComponent(Component comp, Object constraints)** Adds the specified component to this card layout's internal table of names.
2	**void addLayoutComponent(String name, Component comp)** If the layout manager uses a per-component string, adds the component comp to the layout, associating it with the string specified by name.
3	**void first(Container parent)** Flips to the first card of the container.
4	**int getHgap()** Gets the horizontal gap between the components.
5	**float getLayoutAlignmentX(Container parent)** Returns the alignment along the **x** axis.
6	**float getLayoutAlignmentY(Container parent)** Returns the alignment along the **y** axis.
7	**int getVgap()** Gets the vertical gap between the components.
8	**void invalidateLayout(Container target)** Invalidates the layout, indicating that if the layout manager has cached information it should be discarded.
9	**void last(Container parent)** Flips to the last card of the container.
10	**void layoutContainer(Container parent)**

	Lays out the specified container using this card layout.
11	**Dimension maximumLayoutSize(Container target)** Returns the maximum dimensions for this layout given the components in the specified target container.
12	**Dimension minimumLayoutSize(Container parent)** Calculates the minimum size for the specified panel.
13	**void next(Container parent)** Flips to the next card of the specified container.
14	**Dimension preferredLayoutSize(Container parent)** Determines the preferred size of the container argument using this card layout.
15	**void previous(Container parent)** Flips to the previous card of the specified container.
16	**void removeLayoutComponent(Component comp)** Removes the specified component from the layout.
17	**void setHgap(int hgap)** Sets the horizontal gap between the components.
18	**void setVgap(int vgap)** Sets the vertical gap between the components.
19	**void show(Container parent, String name)** Flips to the component that was added to this layout with the specified name, using addLayoutComponent.
20	**String toString()** Returns a string representation of the state of this card layout.

Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

CardLayout Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingLayoutDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingLayoutDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;
   private JLabel msglabel;

   public SwingLayoutDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingLayoutDemo swingLayoutDemo = new SwingLayoutDemo();
      swingLayoutDemo.showCardLayoutDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
```

```
        public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
        }
    });
    controlPanel = new JPanel();
    controlPanel.setLayout(new FlowLayout());

    mainFrame.add(headerLabel);
    mainFrame.add(controlPanel);
    mainFrame.add(statusLabel);
    mainFrame.setVisible(true);
}

private void showCardLayoutDemo(){
    headerLabel.setText("Layout in action: CardLayout");

    final JPanel panel = new JPanel();
    panel.setBackground(Color.CYAN);
    panel.setSize(300,300);

    CardLayout layout = new CardLayout();
    layout.setHgap(10);
    layout.setVgap(10);
    panel.setLayout(layout);

    JPanel buttonPanel = new JPanel(new FlowLayout());

    buttonPanel.add(new JButton("OK"));
    buttonPanel.add(new JButton("Cancel"));

    JPanel textBoxPanel = new JPanel(new FlowLayout());

    textBoxPanel.add(new JLabel("Name:"));
    textBoxPanel.add(new JTextField(20));

    panel.add("Button", buttonPanel);
    panel.add("Text", textBoxPanel);
```

```
      final DefaultComboBoxModel panelName = new DefaultComboBoxModel();

      panelName.addElement("Button");
      panelName.addElement("Text");

      final JComboBox listCombo = new JComboBox(panelName);
      listCombo.setSelectedIndex(0);

      JScrollPane listComboScrollPane = new JScrollPane(listCombo);

      JButton showButton = new JButton("Show");

      showButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            String data = "";
            if (listCombo.getSelectedIndex() != -1) {
               CardLayout cardLayout = (CardLayout)(panel.getLayout());
               cardLayout.show(panel,
               (String)listCombo.getItemAt(listCombo.getSelectedIndex()));
            }
            statusLabel.setText(data);
         }
      });

      controlPanel.add(listComboScrollPane);
      controlPanel.add(showButton);
       controlPanel.add(panel);

      mainFrame.setVisible(true);
   }
}
```

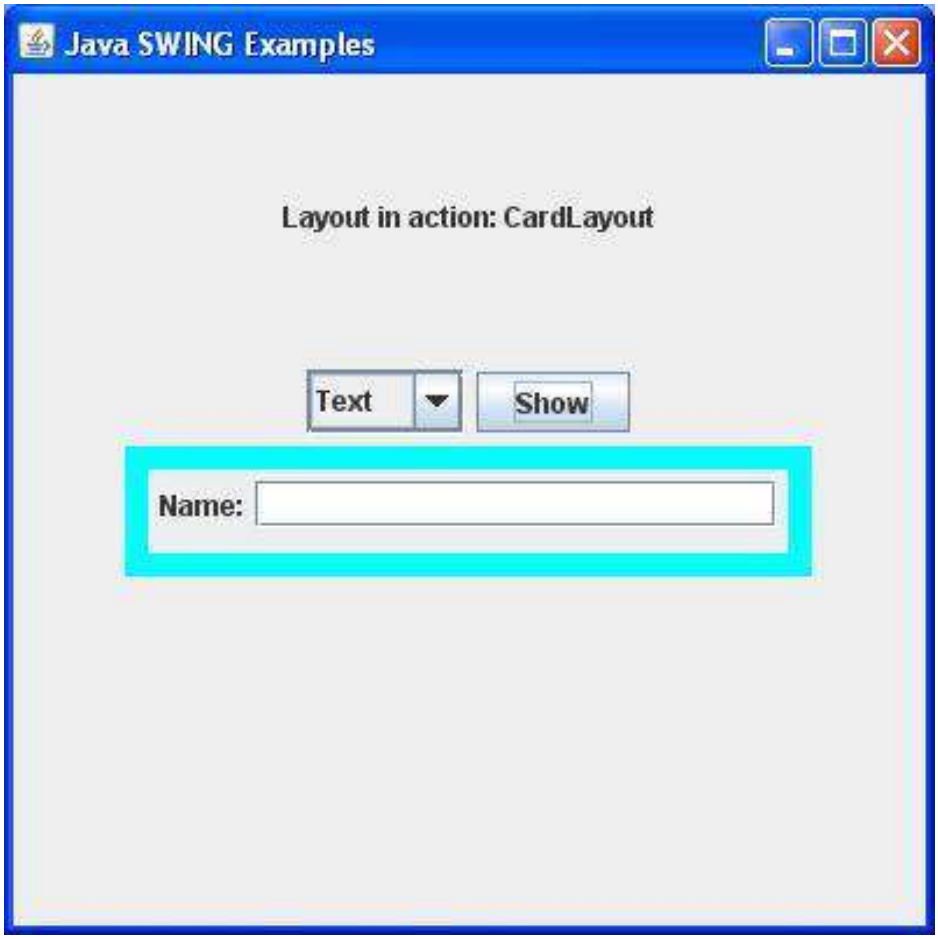
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingLayoutDemo.java
```

If no occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingLayoutDemo
```

Verify the following output.



FlowLayout Class

Introduction

The class **FlowLayout** components in a left-to-right flow.

Class Declaration

Following is the declaration for **java.awt.FlowLayout** class:

```
public class FlowLayout
   extends Object
      implements LayoutManager, Serializable
```

Field

Following are the fields for **java.awt.BorderLayout** class:

- **static int CENTER** - This value indicates that each row of components should be centered.
- **static int LEADING** - This value indicates that each row of components should be justified to the leading edge of the container's orientation. For example, to the left in left-to-right orientations.
- **static int LEFT** - This value indicates that each row of components should be left-justified.
- **static int RIGHT** - This value indicates that each row of components should be right-justified.
- **static int TRAILING** - This value indicates that each row of components should be justified to the trailing edge of the container's orientation. For example, to the right in left-to-right orientations.

Class Constructors

Sr.No.	Constructor & Description
1	**FlowLayout()** Constructs a new FlowLayout with a centered alignment and a default 5-unit horizontal and vertical gap.
2	**FlowLayout(int align)** Constructs a new FlowLayout with the specified alignment and a default 5-unit horizontal and vertical gap.
3	**FlowLayout(int align, int hgap, int vgap)** Creates a new flow layout manager with the indicated alignment and the indicated horizontal and vertical gaps.

Class Methods

Sr.No.	Method & Description
1	**void addLayoutComponent(String name, Component comp)** Adds the specified component to the layout.

2	**int getAlignment()** Gets the alignment for this layout.
3	**int getHgap()** Gets the horizontal gap between the components.
4	**int getVgap()** Gets the vertical gap between the components.
5	**void layoutContainer(Container target)** Lays out the container.
6	**Dimension minimumLayoutSize(Container target)** Returns the minimum dimensions needed to lay out the visible components contained in the specified target container.
7	**Dimension preferredLayoutSize(Container target)** Returns the preferred dimensions for this layout given the visible components in the specified target container.
8	**void removeLayoutComponent(Component comp)** Removes the specified component from the layout.
9	**void setAlignment(int align)** Sets the alignment for this layout.
10	**void setHgap(int hgap)** Sets the horizontal gap between the components.
11	**void setVgap(int vgap)** Sets the vertical gap between the components.
12	**String toString()** Returns a string representation of this FlowLayout object and its values.

Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

FlowLayout Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingLayoutDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingLayoutDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;
   private JLabel msglabel;

   public SwingLayoutDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingLayoutDemo swingLayoutDemo = new SwingLayoutDemo();
      swingLayoutDemo.showFlowLayoutDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);
```

```
      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showFlowLayoutDemo(){
      headerLabel.setText("Layout in action: FlowLayout");

      JPanel panel = new JPanel();
      panel.setBackground(Color.darkGray);
      panel.setSize(200,200);
      FlowLayout layout = new FlowLayout();
      layout.setHgap(10);
      layout.setVgap(10);
      panel.setLayout(layout);
      panel.add(new JButton("OK"));
      panel.add(new JButton("Cancel"));

      controlPanel.add(panel);
      mainFrame.setVisible(true);
   }
}
```

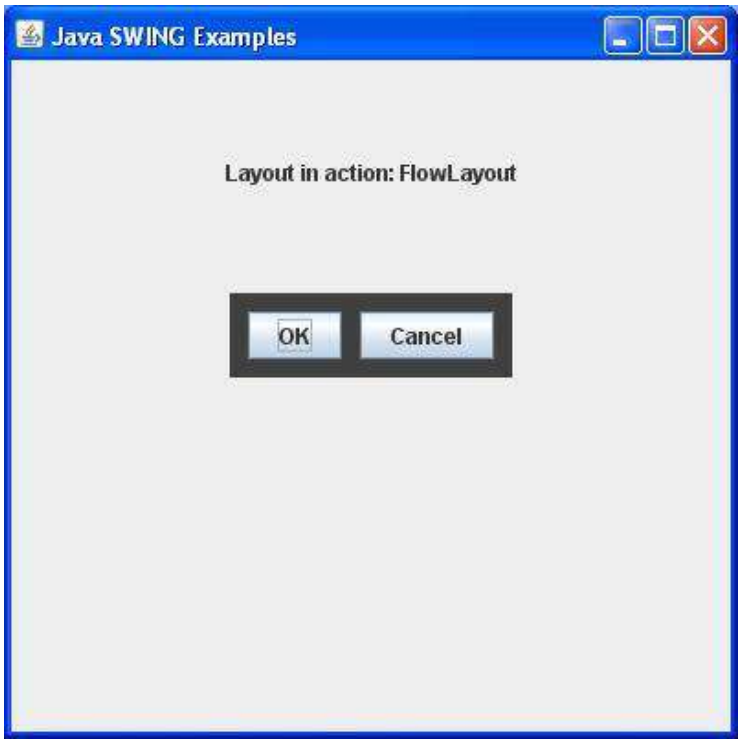
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingLayoutDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingLayoutDemo
```

Verify the following output.



GridLayout Class

Introduction

The class **GridLayout** arranges the components in a rectangular grid.

Class Declaration

Following is the declaration for **java.awt.GridLayout** class:

```
public class GridLayout
   extends Object
      implements LayoutManager, Serializable
```

Class Constructors

Sr.No.	Constructor & Description
1	**GridLayout()** Creates a grid layout with a default of one column per component, in a single row.
2	**GridLayout(int rows, int cols)** Creates a grid layout with the specified number of rows and columns.
3	**GridLayout(int rows, int cols, int hgap, int vgap)** Creates a grid layout with the specified number of rows and columns.

Class Methods

Sr.No.	Method & Description
1	**void addLayoutComponent(String name, Component comp)** Adds the specified component with the specified name to the layout.
2	**int getColumns()** Gets the number of columns in this layout.
3	**int getHgap()** Gets the horizontal gap between the components.
4	**int getRows()** Gets the number of rows in this layout.
5	**int getVgap()** Gets the vertical gap between the components.
6	**void layoutContainer(Container parent)** Lays out the specified container using this layout.
7	**Dimension minimumLayoutSize(Container parent)**

	Determines the minimum size of the container argument using this grid layout.
8	**Dimension preferredLayoutSize(Container parent)** Determines the preferred size of the container argument using this grid layout.
9	**void removeLayoutComponent(Component comp)** Removes the specified component from the layout.
10	**void setColumns(int cols)** Sets the number of columns in this layout to the specified value.
11	**void setHgap(int hgap)** Sets the horizontal gap between the components to the specified value.
12	**void setRows(int rows)** Sets the number of rows in this layout to the specified value.
13	**void setVgap(int vgap)** Sets the vertical gap between the components to the specified value.
14	**String toString()** Returns the string representation of this grid layout's values.

Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

GridLayout Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingLayoutDemo.java

```
package com.tutorialspoint.gui;

import javax.swing.*;

public class SwingLayoutDemo {
```

```
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;
   private JLabel msglabel;

   public SwingLayoutDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingLayoutDemo swingLayoutDemo = new SwingLayoutDemo();
      swingLayoutDemo.showGridLayoutDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }
```

```
   private void showGridLayoutDemo(){
      headerLabel.setText("Layout in action: GridLayout");

      JPanel panel = new JPanel();
      panel.setBackground(Color.darkGray);
      panel.setSize(300,300);
      GridLayout layout = new GridLayout(0,3);
      layout.setHgap(10);
      layout.setVgap(10);

      panel.setLayout(layout);
      panel.add(new JButton("Button 1"));
      panel.add(new JButton("Button 2"));
      panel.add(new JButton("Button 3"));
      panel.add(new JButton("Button 4"));
      panel.add(new JButton("Button 5"));
      controlPanel.add(panel);
      mainFrame.setVisible(true);
   }
}
```

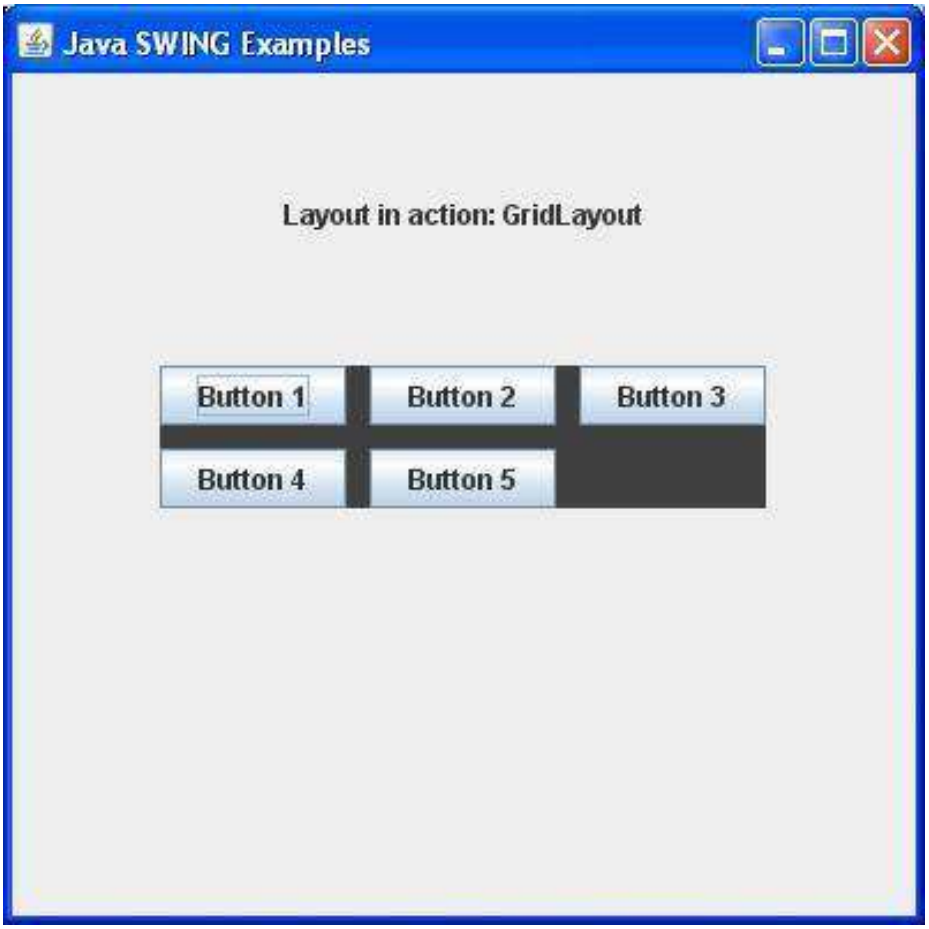
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingLayoutDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingLayoutDemo
```

Verify the following output.



GridBagLayout Class

Introduction

The class **GridBagLayout** arranges the components in a horizontal and vertical manner.

Class Declaration

Following is the declaration for **java.awt.GridBagLayout** class:

```
public class GridBagLayout
   extends Object
      implements LayoutManager2, Serializable
```

Field

Following are the fields for **java.awt.GridBagLayout** class:

- **static int DEFAULT_SIZE** - Indicates the size from the component or the gap should be used for a particular range value.
- **static int PREFERRED_SIZE** - Indicates the preferred size from the component or the gap should be used for a particular range value.

Class Constructors

Sr.No.	Constructor & Description
1	**GridBagLayout()** Creates a grid bag layout manager.

Class Methods

Sr.No.	Method & Description
1	**void addLayoutComponent(Component comp, Object constraints)** Adds the specified component to the layout, using the specified constraints object.
2	**void addLayoutComponent(String name, Component comp)** Adds the specified component with the specified name to the layout.
3	**protected void adjustForGravity(GridBagConstraints constraints, Rectangle r)** Adjusts the **x, y** width and height fields to the correct values depending on the constraint geometry and pads.
4	**protected void AdjustForGravity(GridBagConstraints constraints, Rectangle r)** This method is obsolete and supplied for backwards compatability only; new code should call adjustForGravity instead.
5	**protected void arrangeGrid(Container parent)** Lays out the grid.
6	**protected void ArrangeGrid(Container parent)** This method is obsolete and supplied for backwards compatability only; new code should call arrangeGrid instead.
7	**GridBagConstraints getConstraints(Component comp)**

	Gets the constraints for the specified component.
8	**float getLayoutAlignmentX(Container parent)** Returns the alignment along the **x** axis.
9	**float getLayoutAlignmentY(Container parent)** Returns the alignment along the **y** axis.
10	**int[][] getLayoutDimensions()** Determines column widths and row heights for the layout grid.
11	**protected java.awt.GridBagLayoutInfo getLayoutInfo(Container parent, int sizeflag)** Fills in an instance of GridBagLayoutInfo for the current set of managed children.
12	**protected java.awt.GridBagLayoutInfo GetLayoutInfo(Container parent, int sizeflag)** This method is obsolete and supplied for backwards compatability only; new code should call getLayoutInfo instead.
13	**Point getLayoutOrigin()** Determines the origin of the layout area, in the graphics coordinate space of the target container.
14	**double[][] getLayoutWeights()** Determines the weights of the layout grid's columns and rows.
15	**protected Dimension getMinSize(Container parent, java.awt.GridBagLayoutInfo info)** Figures out the minimum size of the master based on the information from getLayoutInfo().
16	**protected Dimension GetMinSize(Container parent, java.awt.GridBagLayoutInfo info)** This method is obsolete and supplied for backwards compatability only; new code should call getMinSize instead.
17	**void invalidateLayout(Container target)**

	Invalidates the layout, indicating that if the layout manager has cached information it should be discarded.
18	**void layoutContainer(Container parent)** Lays out the specified container using this grid bag layout.
19	**Point location(int x, int y)** Determines which cell in the layout grid contains the point specified by (**x, y**).
20	**protected GridBagConstraints lookupConstraints(Component comp)** Retrieves the constraints for the specified component.
21	**Dimension maximumLayoutSize(Container target)** Returns the maximum dimensions for this layout given the components in the specified target container.
22	**Dimension minimumLayoutSize(Container parent)** Determines the minimum size of the parent container using this grid bag layout.
23	**Dimension preferredLayoutSize(Container parent)** Determines the preferred size of the parent container using this grid bag layout.
24	**void removeLayoutComponent(Component comp)** Removes the specified component from this layout.
25	**void setConstraints(Component comp, GridBagConstraints constraints)** Sets the constraints for the specified component in this layout.
26	**String toString()** Returns a string representation of this grid bag layout's values.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

GridBagLayout Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingLayoutDemo.java

```
package com.tutorialspoint.gui;

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



public class SwingLayoutDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;
   private JLabel msglabel;

   public SwingLayoutDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingLayoutDemo swingLayoutDemo = new SwingLayoutDemo();
      swingLayoutDemo.showGridBagLayoutDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
```

```
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());


      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showGridBagLayoutDemo(){
      headerLabel.setText("Layout in action: GridBagLayout");

      JPanel panel = new JPanel();
      panel.setBackground(Color.darkGray);
      panel.setSize(300,300);
      GridBagLayout layout = new GridBagLayout();

      panel.setLayout(layout);
      GridBagConstraints gbc = new GridBagConstraints();

      gbc.fill = GridBagConstraints.HORIZONTAL;
      gbc.gridx = 0;
      gbc.gridy = 0;
      panel.add(new JButton("Button 1"),gbc);

      gbc.gridx = 1;
      gbc.gridy = 0;
      panel.add(new JButton("Button 2"),gbc);

      gbc.fill = GridBagConstraints.HORIZONTAL;
      gbc.ipady = 20;
      gbc.gridx = 0;
```

```
      gbc.gridy = 1;
      panel.add(new JButton("Button 3"),gbc);



      gbc.gridx = 1;
      gbc.gridy = 1;
      panel.add(new JButton("Button 4"),gbc);

      gbc.gridx = 0;
      gbc.gridy = 2;
      gbc.fill = GridBagConstraints.HORIZONTAL;
      gbc.gridwidth = 2;
      panel.add(new JButton("Button 5"),gbc);

      controlPanel.add(panel);

      mainFrame.setVisible(true);
   }
}
```

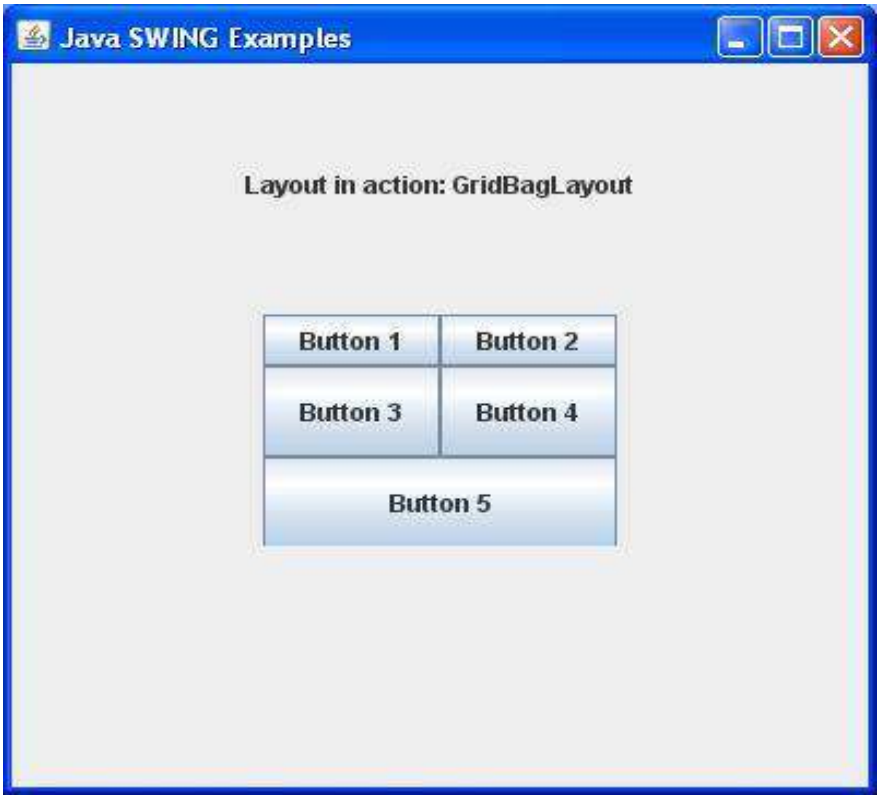
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingLayoutDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingLayoutDemo
```

Verify the following output.



GroupLayout Class

Introduction

The class **GroupLayout** hierarchically groups the components in order to position them in a Container.

Class Declaration

Following is the declaration for **javax.swing.GroupLayout** class:

```
public class GroupLayout
   extends Object
      implements LayoutManager2
```

Field

Following are the fields for **javax.swing.GroupLayout** class:

- **static int DEFAULT_SIZE** - Indicates the size from the component or the gap should be used for a particular range value.
- **static int PREFERRED_SIZE** - Indicates the preferred size from the component or the gap should be used for a particular range value.

Class Constructors

Sr.No.	Constructor & Description
1	**GroupLayout(Container host)** Creates a GroupLayout for the specified Container.

Class Methods

Sr.No.	Method & Description
1	**void addLayoutComponent(Component component, Object constraints)** Notification that a Component has been added to the parent container.
2	**void addLayoutComponent(String name, Component component)** Notification that a Component has been added to the parent container.
3	**GroupLayout.ParallelGroup createBaselineGroup(boolean resizable, boolean anchorBaselineToTop)** Creates and returns a ParallelGroup that aligns its elements along the baseline.
4	**GroupLayout.ParallelGroup createParallelGroup()** Creates and returns a ParallelGroup with an alignment of Alignment.LEADING.
5	**GroupLayout.ParallelGroup createParallelGroup(GroupLayout.Alignment alignment)** Creates and returns a ParallelGroup with the specified alignment.
6	**GroupLayout.ParallelGroup createParallelGroup(GroupLayout.Alignment alignment, boolean resizable)** Creates and returns a ParallelGroup with the specified alignment and resize behavior.
7	**GroupLayout.SequentialGroup createSequentialGroup()** Creates and returns a SequentialGroup.

8	**boolean getAutoCreateContainerGaps()** Returns true if the gaps between the container and the components that border the container are automatically created.
9	**boolean getAutoCreateGaps()** Returns true if the gaps between the components are automatically created.
10	**boolean getHonorsVisibility()** Returns whether the component visiblity is considered when sizing and positioning the components.
11	**float getLayoutAlignmentX(Container parent)** Returns the alignment along the **x** axis.
12	**float getLayoutAlignmentY(Container parent)** Returns the alignment along the **y** axis.
13	**LayoutStyle getLayoutStyle()** Returns the LayoutStyle used for calculating the preferred gap between the components.
14	**void invalidateLayout(Container parent)** Invalidates the layout, indicating that if the layout manager has cached information it should be discarded.
15	**void layoutContainer(Container parent)** Lays out the specified container.
16	**void linkSize(Component... components)** Forces the specified components to have the same size regardless of their preferred, minimum, or maximum sizes.
17	**void linkSize(int axis, Component... components)** Forces the specified components to have the same size along the specified axis regardless of their preferred, minimum, or maximum sizes.
18	**Dimension maximumLayoutSize(Container parent)**

	Returns the maximum size for the specified container.
19	**Dimension minimumLayoutSize(Container parent)** Returns the minimum size for the specified container.
20	**Dimension preferredLayoutSize(Container parent)** Returns the preferred size for the specified container.
21	**void removeLayoutComponent(Component component)** Notifies that a Component has been removed from the parent container.
22	**void replace(Component existingComponent, Component newComponent)** Replaces an existing component with a new one.
23	**void setAutoCreateContainerGaps(boolean autoCreateContainerPadding)** Sets whether a gap between the container and the components that touch the border of the container should automatically be created.
24	**void setAutoCreateGaps(boolean autoCreatePadding)** Sets whether a gap between the components should automatically be created.
25	**void setHonorsVisibility(boolean honorsVisibility)** Sets whether the component visiblity is considered when sizing and positioning the components.
26	**void setHonorsVisibility(Component component, Boolean honorsVisibility)** Sets whether the component's visiblity is considered for sizing and positioning.
27	**void setHorizontalGroup(GroupLayout.Group group)** Sets the Group that positions and sizes the components along the horizontal axis.
28	**void setLayoutStyle(LayoutStyle layoutStyle)** Sets the LayoutStyle used to calculate the preferred gaps between the components.

29	**void setVerticalGroup(GroupLayout.Group group)** Sets the Group that positions and sizes the components along the vertical axis.
30	**String toString()** Returns a string representation of this GroupLayout.

Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

GroupLayout Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingLayoutDemo.java

```
package com.tutorialspoint.gui;

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



public class SwingLayoutDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;
   private JLabel msglabel;

   public SwingLayoutDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingLayoutDemo swingLayoutDemo = new SwingLayoutDemo();
      swingLayoutDemo.showGroupLayoutDemo();
```

```
}

private void prepareGUI(){
   mainFrame = new JFrame("Java SWING Examples");
   mainFrame.setSize(400,400);
   mainFrame.setLayout(new GridLayout(3, 1));

   headerLabel = new JLabel("",JLabel.CENTER );
   statusLabel = new JLabel("",JLabel.CENTER);

   statusLabel.setSize(350,100);
   mainFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent windowEvent){
         System.exit(0);
      }
   });
   controlPanel = new JPanel();
   controlPanel.setLayout(new FlowLayout());


   mainFrame.add(headerLabel);
   mainFrame.add(controlPanel);
   mainFrame.add(statusLabel);
   mainFrame.setVisible(true);
}

private void showGroupLayoutDemo(){
   headerLabel.setText("Layout in action: GroupLayout");

   JPanel panel = new JPanel();
   // panel.setBackground(Color.darkGray);
   panel.setSize(200,200);
   GroupLayout layout = new GroupLayout(panel);
   layout.setAutoCreateGaps(true);
   layout.setAutoCreateContainerGaps(true);
   JButton btn1 = new JButton("Button 1");
   JButton btn2 = new JButton("Button 2");
```

```
      JButton btn3 = new JButton("Button 3");

      layout.setHorizontalGroup(layout.createSequentialGroup()
         .addComponent(btn1)
         .addGroup(layout.createSequentialGroup()
            .addGroup(layout.createParallelGroup(
               GroupLayout.Alignment.LEADING)
               .addComponent(btn2)
               .addComponent(btn3)
            )
         )
      );

      layout.setVerticalGroup(layout.createSequentialGroup()
         .addComponent(btn1)
         .addComponent(btn2)
         .addComponent(btn3)
      );
      panel.setLayout(layout);
      controlPanel.add(panel);

      mainFrame.setVisible(true);
   }
}
```

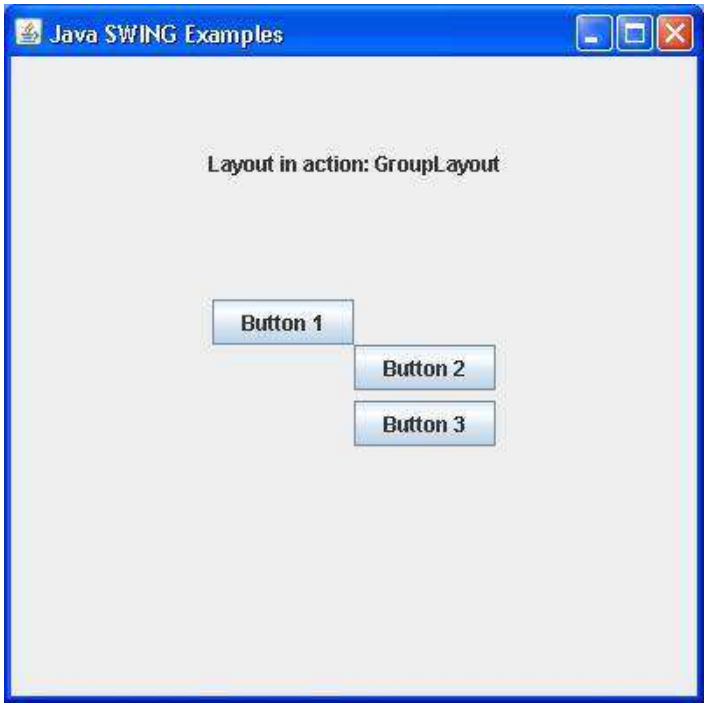
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingLayoutDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingLayoutDemo
```

Verify the following output.



SpringLayout Class

Introduction

The class **SpringLayout** positions the children of its associated container according to a set of constraints.

Class Declaration

Following is the declaration for **javax.swing.SpringLayout** class:

```
public class SpringLayout
   extends Object
      implements LayoutManager2
```

Field

Following are the fields for **javax.swing.SpringLayout** class:

- **static String BASELINE** - Specifies the baseline of a component.
- **static String EAST** - Specifies the right edge of a component's bounding rectangle.
- **static String HEIGHT** - Specifies the height of a component's bounding rectangle.
- **static String HORIZONTAL_CENTER** - Specifies the horizontal center of a component's bounding rectangle.
- **static String NORTH** - Specifies the top edge of a component's bounding rectangle.

- **static String SOUTH** - Specifies the bottom edge of a component's bounding rectangle.
- **static String VERTICAL_CENTER** - Specifies the vertical center of a component's bounding rectangle.
- **static String WEST** - Specifies the left edge of a component's bounding rectangle.
- **static String WIDTH** - Specifies the width of a component's bounding rectangle.

Class Constructors

Sr.No.	Constructor & Description
1	**SpringLayout()** Creates a new SpringLayout.

Class Methods

Sr.No.	Method & Description
1	**void addLayoutComponent(Component component, Object constraints)** If constraints is an instance of SpringLayout.Constraints, associates the constraints with the specified component.
2	**void addLayoutComponent(String name, Component c)** Has no effect, since this layout manager does not use a per-component string.
3	**Spring getConstraint(String edgeName, Component c)** Returns the spring controlling the distance between the specified edge of the component and the top or left edge of its parent.

4	**SpringLayout.Constraints getConstraints(Component c)** Returns the constraints for the specified component.
5	**float getLayoutAlignmentX(Container p)** Returns 0.5f (centered).
6	**float getLayoutAlignmentY(Container p)** Returns 0.5f (centered).
7	**void invalidateLayout(Container p)** Invalidates the layout, indicating that if the layout manager has cached information it should be discarded.
8	**void layoutContainer(Container parent)** Lays out the specified container.
9	**Dimension maximumLayoutSize(Container parent)** Calculates the maximum size dimensions for the specified container, given the components it contains.
10	**Dimension minimumLayoutSize(Container parent)** Calculates the minimum size dimensions for the specified container, given the components it contains.
11	**Dimension preferredLayoutSize(Container parent)** Calculates the preferred size dimensions for the specified container, given the components it contains.
12	**void putConstraint(String e1, Component c1, int pad, String e2, Component c2)** Links edge e1 of component c1 to edge e2 of component c2, with a fixed distance between the edges.
13	**void putConstraint(String e1, Component c1, Spring s, String e2, Component c2)** Links edge e1 of component c1 to edge e2 of component c2.
14	**void removeLayoutComponent(Component c)** Removes the constraints associated with the specified component.

Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

SpringLayout Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingLayoutDemo.java

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

public class SwingSpringLayout {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingSpringLayout(){
      prepareGUI();
   }

   public static void main(String[] args){
   SwingSpringLayout swingLayoutDemo = new SwingSpringLayout();
      swingLayoutDemo.showSpringLayoutDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
```

```
      public void windowClosing(WindowEvent windowEvent){
      System.exit(0);
       }
   });
   controlPanel = new JPanel();
   controlPanel.setLayout(new FlowLayout());

   mainFrame.add(headerLabel);
   mainFrame.add(controlPanel);
   mainFrame.add(statusLabel);
   mainFrame.setVisible(true);
}

private void showSpringLayoutDemo(){

   headerLabel.setText("Layout in action: SpringLayout");
   SpringLayout layout = new SpringLayout();

   JPanel panel = new JPanel();
   panel.setLayout(layout);
   JLabel label = new JLabel("Enter Name: ");
   JTextField textField = new JTextField("", 15);
   panel.add(label);
   panel.add(textField);

   layout.putConstraint(SpringLayout.WEST, label,5,
      SpringLayout.WEST, controlPanel);
   layout.putConstraint(SpringLayout.NORTH, label,5,
      SpringLayout.NORTH, controlPanel);
   layout.putConstraint(SpringLayout.WEST, textField,5,
      SpringLayout.EAST, label);
   layout.putConstraint(SpringLayout.NORTH, textField,5,
      SpringLayout.NORTH, controlPanel);
   layout.putConstraint(SpringLayout.EAST, panel,5,
      SpringLayout.EAST, textField);
   layout.putConstraint(SpringLayout.SOUTH, panel,5,
      SpringLayout.SOUTH, textField);
```

```
      controlPanel.add(panel);
      mainFrame.setVisible(true);
   }
}
```

Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingLayoutDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

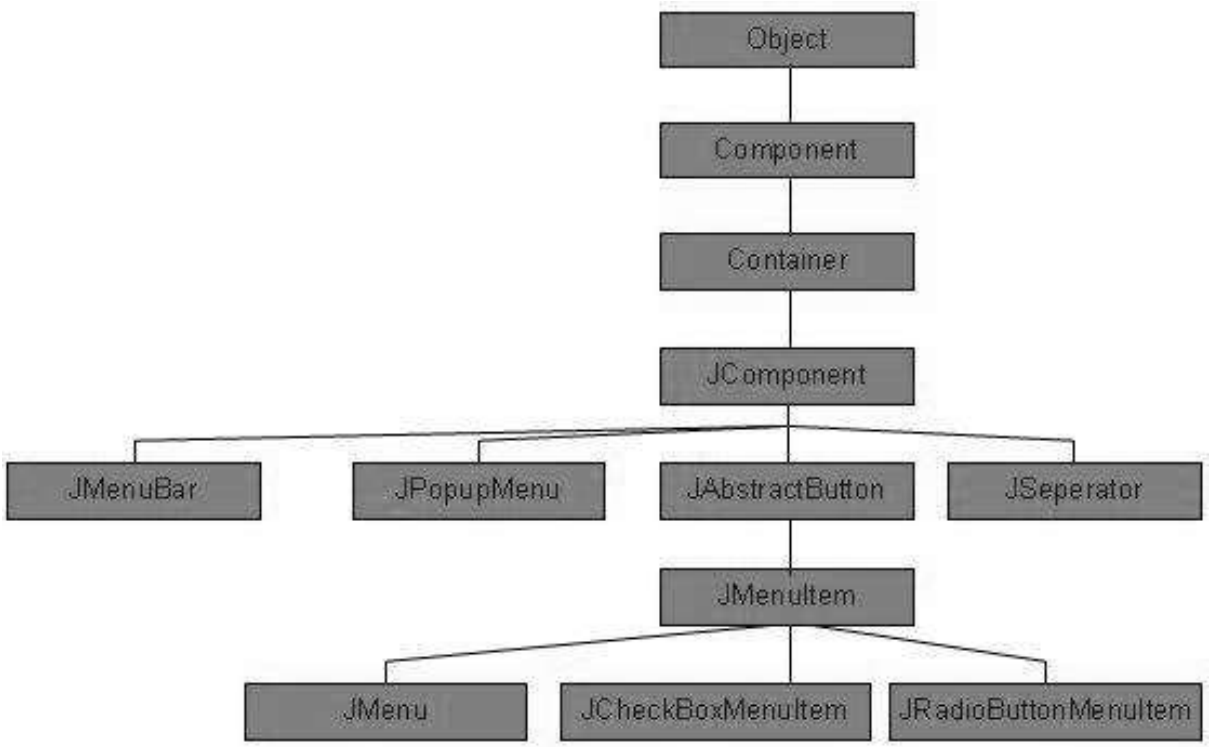
```
D:\SWING>java com.tutorialspoint.gui.SwingLayoutDemo
```

Verify the following output.

9. Swing – Menu Classes

As we know that every top-level window has a menu bar associated with it. This menu bar consists of various menu choices available to the end user. Further, each choice contains a list of options, which is called drop-down menus. Menu and MenuItem controls are subclass of MenuComponent class.

Menu Hiearchy



Menu Controls

Sr. No.	Control & Description
1	**JMenuBar** The JMenuBar object is associated with the top-level window.
2	**JMenuItem** The items in the menu must belong to the JMenuItem or any of its subclass.
3	**JMenu** The JMenu object is a pull-down menu component which is displayed from the menu bar.

4	**JCheckboxMenuItem** JCheckboxMenuItem is the subclass of JMenuItem.
5	**JRadioButtonMenuItem** JRadioButtonMenuItem is the subclass of JMenuItem.
6	**JPopupMenu** JPopupMenu can be dynamically popped up at a specified position within a component.

JMenuBar Class

Introduction

The JMenuBar class provides an implementation of a menu bar.

Class Declaration

Following is the declaration for **javax.swing.JMenuBar** class:

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

Class Constructors

Sr.No.	Constructor & Description
1	**JMenuBar()** Creates a new menu bar.

Class Methods

Sr.No.	Method & Description
1	**JMenu add(JMenu c)** Appends the specified menu to the end of the menu bar.
2	**void addNotify()**

	Overrides JComponent.addNotify to register this menu bar with the current keyboard manager.
3	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JMenuBar.
4	**Component getComponent()** Implemented to be a MenuElement.
5	**Component getComponentAtIndex(int i)** Deprecated. Replaced by getComponent(int i)
6	**int getComponentIndex(Component c)** Returns the index of the specified component.
7	**JMenu getHelpMenu()** Gets the help menu for the menu bar.
8	**Insets getMargin()** Returns the margin between the menubar's border and its menus.
9	**JMenu getMenu(int index)** Returns the menu at the specified position in the menu bar.
10	**int getMenuCount()** Returns the number of items in the menu bar.
11	**SingleSelectionModel getSelectionModel()** Returns the model object that handles single selections.
12	**MenuElement[] getSubElements()** Implemented to be a MenuElement. Returns the menus in this menu bar.
13	**MenuBarUI getUI()** Returns the menubar's current UI.
14	**String getUIClassID()** Returns the name of the L&F class that renders this component.
15	

	boolean isBorderPainted() Returns true if the menu bars border should be painted.
16	**boolean isSelected()** Returns true if the menu bar currently has a component selected.
17	**void menuSelectionChanged(boolean isIncluded)** Implemented to be a MenuElement, does nothing.
18	**protected void paintBorder(Graphics g)** Paints the menubar's border, if the BorderPainted property is true.
19	**protected String paramString()** Returns a string representation of this JMenuBar.
20	**protected boolean processKeyBinding(KeyStroke ks, KeyEvent e, int condition, boolean pressed)** Subclassed to check all the child menus.
21	**void processKeyEvent(KeyEvent e, MenuElement[] path, MenuSelectionManager manager)** Implemented to be a MenuElement, does nothing.
22	**void processMouseEvent(MouseEvent event, MenuElement[] path, MenuSelectionManager manager)** Implemented to be a MenuElement, does nothing.
23	**void removeNotify()** Overrides JComponent.removeNotify to unregister this menu bar with the current keyboard manager.
24	**void setBorderPainted(boolean b)** Sets whether the border should be painted.
25	**void setHelpMenu(JMenu menu)** Sets the help menu that appears when the user selects the "help" option in the menu bar.
27	**void setMargin(Insets m)**

	Sets the margin between the menubar's border and its menus.
28	**void setSelected(Component sel)** Sets the currently selected component, producing a change to the selection model.
29	**void setSelectionModel(SingleSelectionModel model)** Sets the model object to handle single selections.
30	**void setUI(MenuBarUI ui)** Sets the L&F object that renders this component.
31	**void updateUI()** Resets the UI property with a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JMenuBar Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingMenuDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingMenuDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;
```

```
public SwingMenuDemo(){
   prepareGUI();
}

public static void main(String[] args){
   SwingMenuDemo  swingMenuDemo = new SwingMenuDemo();
   swingMenuDemo.showMenuDemo();
}

private void prepareGUI(){
   mainFrame = new JFrame("Java SWING Examples");
   mainFrame.setSize(400,400);
   mainFrame.setLayout(new GridLayout(3, 1));

   headerLabel = new JLabel("",JLabel.CENTER );
   statusLabel = new JLabel("",JLabel.CENTER);

   statusLabel.setSize(350,100);
   mainFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent windowEvent){
         System.exit(0);
      }
   });
   controlPanel = new JPanel();
   controlPanel.setLayout(new FlowLayout());

   mainFrame.add(headerLabel);
   mainFrame.add(controlPanel);
   mainFrame.add(statusLabel);
   mainFrame.setVisible(true);
}
```

```
private void showMenuDemo(){
   //create a menu bar
   final JMenuBar menuBar = new JMenuBar();

   //create menus
   JMenu fileMenu = new JMenu("File");
   JMenu editMenu = new JMenu("Edit");
   final JMenu aboutMenu = new JMenu("About");
   final JMenu linkMenu = new JMenu("Links");

   //create menu items
   JMenuItem newMenuItem = new JMenuItem("New");
   newMenuItem.setMnemonic(KeyEvent.VK_N);
   newMenuItem.setActionCommand("New");

   JMenuItem openMenuItem = new JMenuItem("Open");
   openMenuItem.setActionCommand("Open");

   JMenuItem saveMenuItem = new JMenuItem("Save");
   saveMenuItem.setActionCommand("Save");

   JMenuItem exitMenuItem = new JMenuItem("Exit");
   exitMenuItem.setActionCommand("Exit");

   JMenuItem cutMenuItem = new JMenuItem("Cut");
   cutMenuItem.setActionCommand("Cut");

   JMenuItem copyMenuItem = new JMenuItem("Copy");
   copyMenuItem.setActionCommand("Copy");

   JMenuItem pasteMenuItem = new JMenuItem("Paste");
   pasteMenuItem.setActionCommand("Paste");

   MenuItemListener menuItemListener = new MenuItemListener();
```

```
      newMenuItem.addActionListener(menuItemListener);
      openMenuItem.addActionListener(menuItemListener);
      saveMenuItem.addActionListener(menuItemListener);
      exitMenuItem.addActionListener(menuItemListener);
      cutMenuItem.addActionListener(menuItemListener);
      copyMenuItem.addActionListener(menuItemListener);
      pasteMenuItem.addActionListener(menuItemListener);

      final JCheckBoxMenuItem showWindowMenu = new JCheckBoxMenuItem("Show
About", true);
      showWindowMenu.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
            if(showWindowMenu.getState()){
               menuBar.add(aboutMenu);
            }else{
               menuBar.remove(aboutMenu);
            }
         }
      });

      final JRadioButtonMenuItem showLinksMenu =
         new JRadioButtonMenuItem("Show Links", true);
      showLinksMenu.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
            if(menuBar.getMenu(3)!= null){
               menuBar.remove(linkMenu);
               mainFrame.repaint();
            }else{
               menuBar.add(linkMenu);
               mainFrame.repaint();
            }
         }
      });


      //add menu items to menus
      fileMenu.add(newMenuItem);
```

```
      fileMenu.add(openMenuItem);
      fileMenu.add(saveMenuItem);
      fileMenu.addSeparator();
      fileMenu.add(showWindowMenu);
      fileMenu.addSeparator();
      fileMenu.add(showLinksMenu);
      fileMenu.addSeparator();
      fileMenu.add(exitMenuItem);
      editMenu.add(cutMenuItem);
      editMenu.add(copyMenuItem);
      editMenu.add(pasteMenuItem);

      //add menu to menubar
      menuBar.add(fileMenu);
      menuBar.add(editMenu);
      menuBar.add(aboutMenu);
      menuBar.add(linkMenu);

      //add menubar to the frame
      mainFrame.setJMenuBar(menuBar);
      mainFrame.setVisible(true);
   }

   class MenuItemListener implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         statusLabel.setText(e.getActionCommand()
         + " JMenuItem clicked.");
      }
   }
}
```

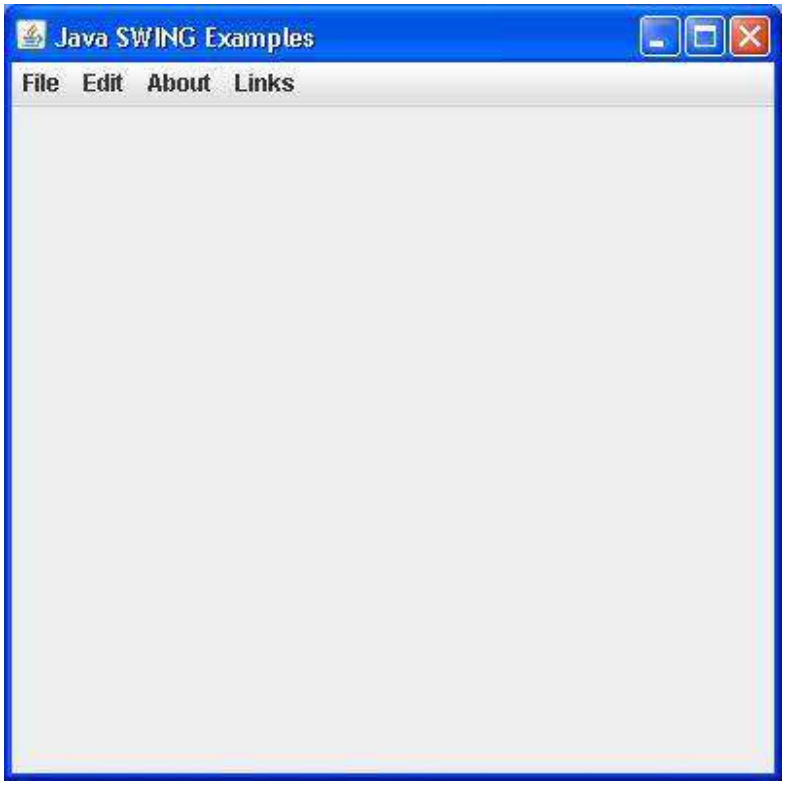
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingMenuDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingMenuDemo
```

Verify the following output.



JMenuItem Class

Introduction

The JMenuItem class represents the actual item in a menu. All items in a menu should derive from class JMenuItem, or one of its subclasses. By default, it embodies a simple labeled menu item.

Class Declaration

Following is the declaration for **javax.swing.JMenuItem** class:

```
public class JMenuItem
   extends AbstractButton
      implements Accessible, MenuElement
```

Class Constructors

Sr.No.	Constructor & Description
1	**JMenuItem()** Creates a JMenuItem with no set text or icon.
2	**JMenuItem(Action a)** Creates a menu item whose properties are taken from the specified Action.
3	**JMenuItem(Icon icon)** Creates a JMenuItem with the specified icon.
4	**JMenuItem(String text)** Creates a JMenuItem with the specified text.
5	**JMenuItem(String text, Icon icon)** Creates a JMenuItem with the specified text and icon.
6	**JMenuItem(String text, int mnemonic)** Creates a JMenuItem with the specified text and keyboard mnemonic.

Class Methods

Sr.No.	Method & Description
1	**protected void actionPropertyChanged(Action action, String propertyName)** Updates the button's state in response to property changes in the associated action.
2	**void addMenuDragMouseListener(MenuDragMouseListener l)** Adds a MenuDragMouseListener to the menu item.
3	**void addMenuKeyListener(MenuKeyListener l)** Adds a MenuKeyListener to the menu item.

4	**protected void configurePropertiesFromAction(Action a)** Sets the properties on this button to match those in the specified Action.
5	**protected void fireMenuDragMouseDragged(MenuDragMouseEvent event)** Notifies all listeners that have registered interest for notification on this event type.
6	**protected void fireMenuDragMouseEntered(MenuDragMouseEvent event)** Notifies all listeners that have registered interest for notification on this event type.
7	**protected void fireMenuDragMouseExited(MenuDragMouseEvent event)** Notifies all listeners that have registered interest for notification on this event type.
8	**protected void fireMenuDragMouseReleased(MenuDragMouseEvent event)** Notifies all listeners that have registered interest for notification on this event type.
9	**protected void fireMenuKeyPressed(MenuKeyEvent event)** Notifies all listeners that have registered interest for notification on this event type.
10	**protected void fireMenuKeyReleased(MenuKeyEvent event)** Notifies all listeners that have registered interest for notification on this event type.
11	**protected void fireMenuKeyTyped(MenuKeyEvent event)** Notifies all listeners that have registered interest for notification on this event type.
12	**KeyStroke getAccelerator()** Returns the KeyStroke which serves as an accelerator for the menu item.
13	

	AccessibleContext getAccessibleContext() Returns the AccessibleContext associated with this JMenuItem.
14	**Component getComponent()** Returns the java.awt.Component used to paint this object.
15	**MenuDragMouseListener[] getMenuDragMouseListeners()** Returns an array of all the MenuDragMouseListeners added to this JMenuItem with addMenuDragMouseListener().
16	**MenuKeyListener[] getMenuKeyListeners()** Returns an array of all the MenuKeyListeners added to this JMenuItem with addMenuKeyListener().
17	**MenuElement[] getSubElements()** This method returns an array containing the sub-menu components for this menu component.
18	**String getUIClassID()** Returns the suffix used to construct the name of the L&F class, used to render this component.
19	**protected void init(String text, Icon icon)** Initializes the menu item with the specified text and icon.
20	**boolean isArmed()** Returns whether the menu item is "armed".
21	**void menuSelectionChanged(boolean isIncluded)** Called by the MenuSelectionManager when the MenuElement is selected or unselected.

22	**protected String paramString()** Returns a string representation of this JMenuItem.
23	**void processKeyEvent(KeyEvent e, MenuElement[] path, MenuSelectionManager manager)** Processes a key event forwarded from the MenuSelectionManager and changes the menu selection, if necessary, by using MenuSelectionManager's API.
24	**void processMenuDragMouseEvent(MenuDragMouseEvent e)** Handles mouse drag in a menu.
25	**void processMenuKeyEvent(MenuKeyEvent e)** Handles a keystroke in a menu.
26	**void processMouseEvent(MouseEvent e, MenuElement[] path, MenuSelectionManager manager)** Processes a mouse event forwarded from the MenuSelectionManager and changes the menu selection, if necessary, by using the MenuSelectionManager's API.
27	**void removeMenuDragMouseListener(MenuDragMouseListener l)** Removes a MenuDragMouseListener from the menu item.
28	**void removeMenuKeyListener(MenuKeyListener l)** Removes a MenuKeyListener from the menu item.
29	**void setAccelerator(KeyStroke keyStroke)** Sets the key combination which invokes the menu item's action listeners without navigating the menu hierarchy.
30	**void setArmed(boolean b)** Identifies the menu item as "armed".
31	**void setEnabled(boolean b)** Enables or disables the menu item.

32	**void setModel(ButtonModel newModel)** Sets the model that this button represents.
33	**void setUI(MenuItemUI ui)** Sets the look and feel object that renders this component.
34	**void updateUI()** Resets the UI property with a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.JAbstractButton
- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JMenuItem Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingMenuDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingMenuDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingMenuDemo(){
      prepareGUI();
   }
```

```
   public static void main(String[] args){
      SwingMenuDemo  swingMenuDemo = new SwingMenuDemo();
      swingMenuDemo.showMenuDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showMenuDemo(){
      //create a menu bar
      final JMenuBar menuBar = new JMenuBar();

      //create menus
      JMenu fileMenu = new JMenu("File");
      JMenu editMenu = new JMenu("Edit");
      final JMenu aboutMenu = new JMenu("About");
      final JMenu linkMenu = new JMenu("Links");
```

```
//create menu items
JMenuItem newMenuItem = new JMenuItem("New");
newMenuItem.setMnemonic(KeyEvent.VK_N);
newMenuItem.setActionCommand("New");

JMenuItem openMenuItem = new JMenuItem("Open");
openMenuItem.setActionCommand("Open");



JMenuItem saveMenuItem = new JMenuItem("Save");
saveMenuItem.setActionCommand("Save");

JMenuItem exitMenuItem = new JMenuItem("Exit");
exitMenuItem.setActionCommand("Exit");

JMenuItem cutMenuItem = new JMenuItem("Cut");
cutMenuItem.setActionCommand("Cut");

JMenuItem copyMenuItem = new JMenuItem("Copy");
copyMenuItem.setActionCommand("Copy");

JMenuItem pasteMenuItem = new JMenuItem("Paste");
pasteMenuItem.setActionCommand("Paste");

MenuItemListener menuItemListener = new MenuItemListener();

newMenuItem.addActionListener(menuItemListener);
openMenuItem.addActionListener(menuItemListener);
saveMenuItem.addActionListener(menuItemListener);
exitMenuItem.addActionListener(menuItemListener);
cutMenuItem.addActionListener(menuItemListener);
copyMenuItem.addActionListener(menuItemListener);
pasteMenuItem.addActionListener(menuItemListener);

final JCheckBoxMenuItem showWindowMenu =
```

```
      new JCheckBoxMenuItem("Show About", true);
   showWindowMenu.addItemListener(new ItemListener() {
      public void itemStateChanged(ItemEvent e) {
         if(showWindowMenu.getState()){
            menuBar.add(aboutMenu);
         }else{
            menuBar.remove(aboutMenu);
         }

      }
   });

   final JRadioButtonMenuItem showLinksMenu =
      new JRadioButtonMenuItem("Show Links", true);
   showLinksMenu.addItemListener(new ItemListener() {
      public void itemStateChanged(ItemEvent e) {
         if(menuBar.getMenu(3)!= null){
            menuBar.remove(linkMenu);
            mainFrame.repaint();
         }else{
            menuBar.add(linkMenu);
            mainFrame.repaint();
         }
      }
   });

   //add menu items to menus
   fileMenu.add(newMenuItem);
   fileMenu.add(openMenuItem);
   fileMenu.add(saveMenuItem);
   fileMenu.addSeparator();
   fileMenu.add(showWindowMenu);
   fileMenu.addSeparator();
   fileMenu.add(showLinksMenu);
   fileMenu.addSeparator();
   fileMenu.add(exitMenuItem);
   editMenu.add(cutMenuItem);
```

```
      editMenu.add(copyMenuItem);
      editMenu.add(pasteMenuItem);

      //add menu to menubar
      menuBar.add(fileMenu);
      menuBar.add(editMenu);
      menuBar.add(aboutMenu);
      menuBar.add(linkMenu);

      //add menubar to the frame
      mainFrame.setJMenuBar(menuBar);
      mainFrame.setVisible(true);
   }

   class MenuItemListener implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         statusLabel.setText(e.getActionCommand()
         + " JMenuItem clicked.");
      }
   }
}
```

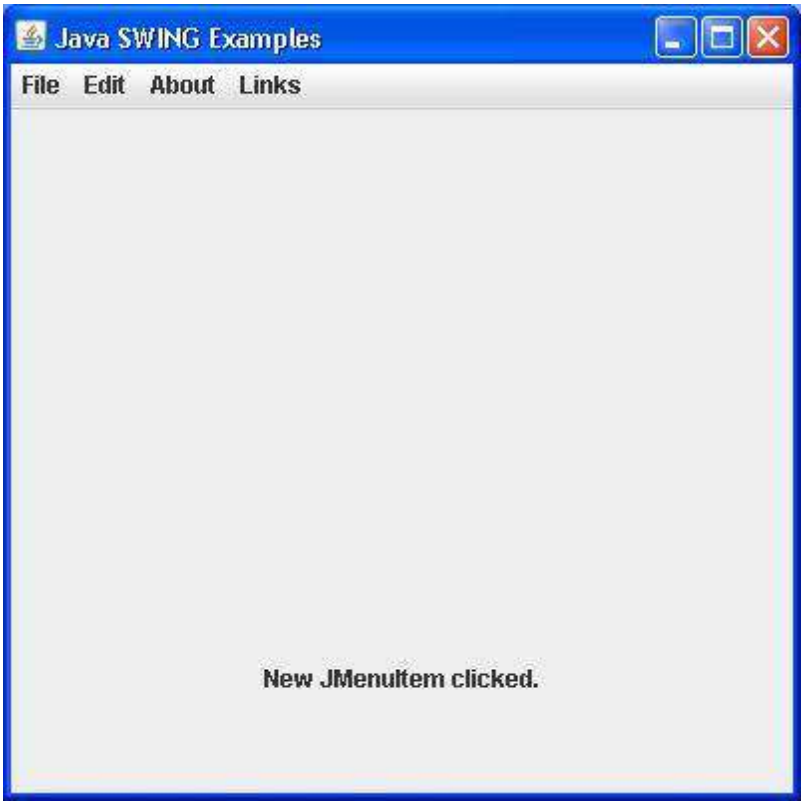
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingMenuDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingMenuDemo
```

Verify the following output. (Click on File Menu. Select any menu item.)



JMenu Class

Introduction

The Menu class represents the pull-down menu component which is deployed from a menu bar.

Class Declaration

Following is the declaration for **javax.swing.JMenu** class:

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

Field

Following is the field for **java.awt.Component** class:

- **protected JMenu.WinListener popupListener** - The window-closing listener for the popup.

Class Constructors

Sr.No.	Constructor & Description
1	**JMenu()** Constructs a new JMenu with no text.
2	**JMenu(Action a)** Constructs a menu whose properties are taken from the Action supplied.
3	**JMenu(String s)** Constructs a new JMenu with the supplied string as its text.
4	**JMenu(String s, boolean b)** Constructs a new JMenu with the supplied string as its text and specified as a tear-off menu or not.

Class Methods

Sr.No.	Method & Description
1	**JMenuItem add(Action a)** Creates a new menu item attached to the specified Action object and appends it to the end of this menu.
2	**Component add(Component c)** Appends a component to the end of this menu.
3	**Component add(Component c, int index)** Adds the specified component to this container at the given position.
4	**JMenuItem add(JMenuItem menuItem)** Appends a menu item to the end of this menu.
5	**JMenuItem add(String s)** Creates a new menu item with the specified text and appends it to the end of this menu.

6	**void addMenuListener(MenuListener l)** Adds a listener for menu events.
7	**void addSeparator()** Appends a new separator to the end of the menu.
8	**void applyComponentOrientation(ComponentOrientation o)** Sets the ComponentOrientation property of this menu and all components contained within it.
9	**protected PropertyChangeListener createActionChangeListener(JMenuItem b)** Returns a properly configured PropertyChangeListener which updates the control as changes to the Action occur.
10	**protected JMenuItem createActionComponent(Action a)** Factory method which creates the JMenuItem for Actions added to the JMenu.
11	**protected JMenu.WinListener createWinListener(JPopupMenu p)** Creates a window-closing listener for the popup.
12	**void doClick(int pressTime)** Programmatically performs a "click".
13	**protected void fireMenuCanceled()** Notifies all listeners that have registered interest for notification on this event type.
14	**protected void fireMenuDeselected()** Notifies all listeners that have registered interest for notification on this event type.
15	**protected void fireMenuSelected()** Notifies all listeners that have registered interest for notification on this event type.
16	**AccessibleContext getAccessibleContext()**

	Gets the AccessibleContext associated with this JMenu.
17	**Component getComponent()** Returns the java.awt.Component used to paint this MenuElement.
18	**int getDelay()** Returns the suggested delay, in milliseconds, before submenus are popped up or down.
19	**JMenuItem getItem(int pos)** Returns the JMenuItem at the specified position.
20	**int getItemCount()** Returns the number of items on the menu, including separators.
21	**Component getMenuComponent(int n)** Returns the component at position **n**.
22	**int getMenuComponentCount()** Returns the number of components on the menu.
23	**Component[] getMenuComponents()** Returns an array of Components of the menu's subcomponents.
24	**MenuListener[]getMenuListeners()** Returns an array of all the MenuListeners added to this JMenu with addMenuListener().
25	**JPopupMenu getPopupMenu()** Returns the popupmenu associated with this menu.
26	**protected Point getPopupMenuOrigin()** Computes the origin for the JMenu's popup menu.
27	**MenuElement[] getSubElements()** Returns an array of MenuElements containing the submenu for this menu component.
28	

	String getUIClassID() Returns the name of the L&F class that renders this component.
29	**JMenuItem insert(Action a, int pos)** Inserts a new menu item attached to the specified Action object at a given position.
30	**JMenuItem insert(JMenuItem mi, int pos)** Inserts the specified JMenuitem at a given position.
31	**void insert(String s, int pos)** Inserts a new menu item with the specified text at a given position.
32	**void insertSeparator(int index)** Inserts a separator at the specified position.
33	**boolean isMenuComponent(Component c)** Returns true, if the specified component exists in the submenu hierarchy.
34	**boolean isPopupMenuVisible()** Returns true, if the menu's popup window is visible.
35	**boolean isSelected()** Returns true. if the menu is currently selected (highlighted).
36	**boolean isTearOff()** Returns true, if the menu can be torn off.
37	**boolean isTopLevelMenu()** Returns true, if the menu is a 'top-level menu', that is, if it is the direct child of a menubar.
38	**void menuSelectionChanged(boolean isIncluded)** Messaged when the menubar selection changes to activate or deactivate this menu.
39	**protected String paramString()**

	Returns a string representation of this JMenu.
40	**protected void processKeyEvent(KeyEvent evt)** Processes key stroke events such as mnemonics and accelerators.
41	**void remove(Component c)** Removes the component **c** from this menu.
42	**void remove(int pos)** Removes the menu item at the specified index from this menu.
43	**void remove(JMenuItem item)** Removes the specified menu item from this menu.
44	**void removeAll()** Removes all menu items from this menu.
45	**void removeMenuListener(MenuListener l)** Removes a listener for menu events.
46	**void setAccelerator(KeyStroke keyStroke)** setAccelerator is not defined for JMenu.
47	**void setComponentOrientation(ComponentOrientation o)** Sets the language-sensitive orientation that is used to order the elements or text within this component.
48	**void setDelay(int d)** Sets the suggested delay before the menu's PopupMenu is popped up or down.
49	**void setMenuLocation(int x, int y)** Sets the location of the popup component.
50	**void setModel(ButtonModel newModel)** Sets the data model for the "menu button" - the label that the user clicks to open or close the menu.
51	

	void setPopupMenuVisible(boolean b) Sets the visibility of the menu's popup.
52	**void setSelected(boolean b)** Sets the selection status of the menu.
53	**void updateUI()** Resets the UI property with a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.JAbstractButton
- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JMenu Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingMenuDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingMenuDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingMenuDemo(){
      prepareGUI();
   }
```

```
   public static void main(String[] args){
      SwingMenuDemo  swingMenuDemo = new SwingMenuDemo();
      swingMenuDemo.showMenuDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showMenuDemo(){
      //create a menu bar
      final JMenuBar menuBar = new JMenuBar();

      //create menus
      JMenu fileMenu = new JMenu("File");
      JMenu editMenu = new JMenu("Edit");
      final JMenu aboutMenu = new JMenu("About");
      final JMenu linkMenu = new JMenu("Links");
```

```
//create menu items
JMenuItem newMenuItem = new JMenuItem("New");
newMenuItem.setMnemonic(KeyEvent.VK_N);
newMenuItem.setActionCommand("New");

JMenuItem openMenuItem = new JMenuItem("Open");
openMenuItem.setActionCommand("Open");




JMenuItem saveMenuItem = new JMenuItem("Save");
saveMenuItem.setActionCommand("Save");

JMenuItem exitMenuItem = new JMenuItem("Exit");
exitMenuItem.setActionCommand("Exit");

JMenuItem cutMenuItem = new JMenuItem("Cut");
cutMenuItem.setActionCommand("Cut");

JMenuItem copyMenuItem = new JMenuItem("Copy");
copyMenuItem.setActionCommand("Copy");

JMenuItem pasteMenuItem = new JMenuItem("Paste");
pasteMenuItem.setActionCommand("Paste");

MenuItemListener menuItemListener = new MenuItemListener();

newMenuItem.addActionListener(menuItemListener);
openMenuItem.addActionListener(menuItemListener);
saveMenuItem.addActionListener(menuItemListener);
exitMenuItem.addActionListener(menuItemListener);
cutMenuItem.addActionListener(menuItemListener);
copyMenuItem.addActionListener(menuItemListener);
pasteMenuItem.addActionListener(menuItemListener);
```

```
final JCheckBoxMenuItem showWindowMenu =
   new JCheckBoxMenuItem("Show About", true);
showWindowMenu.addItemListener(new ItemListener() {
   public void itemStateChanged(ItemEvent e) {
      if(showWindowMenu.getState()){
         menuBar.add(aboutMenu);
      }else{

   menuBar.remove(aboutMenu);
      }
   }
});

final JRadioButtonMenuItem showLinksMenu =
   new JRadioButtonMenuItem("Show Links", true);
showLinksMenu.addItemListener(new ItemListener() {
   public void itemStateChanged(ItemEvent e) {
      if(menuBar.getMenu(3)!= null){
         menuBar.remove(linkMenu);
         mainFrame.repaint();
      }else{
         menuBar.add(linkMenu);
         mainFrame.repaint();
      }
   }
});

//add menu items to menus
fileMenu.add(newMenuItem);
fileMenu.add(openMenuItem);
fileMenu.add(saveMenuItem);
fileMenu.addSeparator();
fileMenu.add(showWindowMenu);
fileMenu.addSeparator();
fileMenu.add(showLinksMenu);
fileMenu.addSeparator();
fileMenu.add(exitMenuItem);
```

```
      editMenu.add(cutMenuItem);
      editMenu.add(copyMenuItem);
      editMenu.add(pasteMenuItem);

      //add menu to menubar
      menuBar.add(fileMenu);
      menuBar.add(editMenu);
      menuBar.add(aboutMenu);
      menuBar.add(linkMenu);

      //add menubar to the frame
      mainFrame.setJMenuBar(menuBar);
      mainFrame.setVisible(true);
   }

   class MenuItemListener implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         statusLabel.setText(e.getActionCommand()
         + " JMenuItem clicked.");
      }
   }
}
```

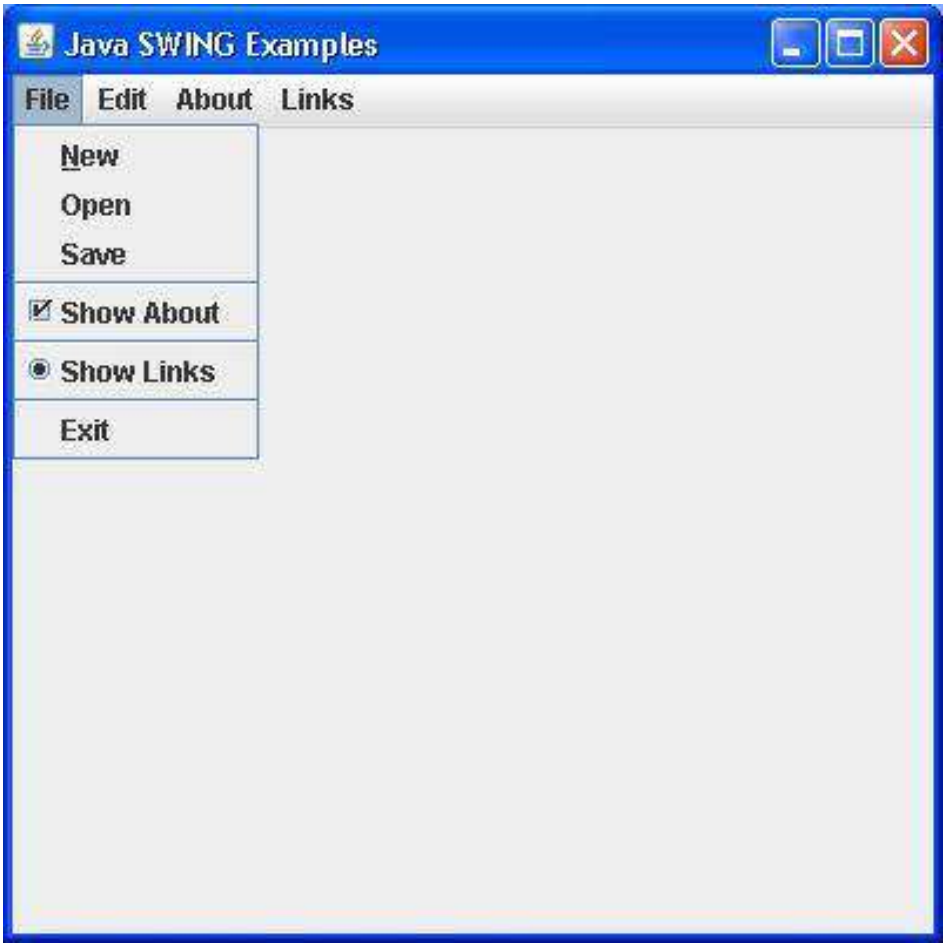
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingMenuDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingMenuDemo
```

Verify the following output. (Click on File Menu.)



JCheckboxMenuItem Class

Introduction

The JCheckboxMenuItem class represents a checkbox which can be included in a menu. Selecting the checkbox in the menu changes the control's state from **on** to **off** or from **off** to **on**.

Class Declaration

Following is the declaration for **javax.swing.JCheckBoxMenuItem** class:

```
public class JCheckBoxMenuItem
   extends JMenuItem
      implements SwingConstants, Accessible
```

Class Constructors

Sr.No.	Constructor & Description
1	**JCheckboxMenuItem()** Creates an initially unselected checkbox menu item with no set text or icon.
2	**JCheckboxMenuItem(Action a)** Creates a menu item whose properties are taken from the Action supplied.
3	**JCheckboxMenuItem(Icon icon)** Creates an initially unselected checkbox menu item with an icon.
4	**JCheckboxMenuItem(String text)** Creates an initially unselected checkbox menu item with text.
5	**JCheckboxMenuItem(String text, boolean b)** Creates a checkbox menu item with the specified text and selection state.
6	**JCheckboxMenuItem(String text, Icon icon)** Creates an initially unselected checkbox menu item with the specified text and icon.
7	**JCheckboxMenuItem(String text, Icon icon, boolean b)** Creates a checkbox menu item with the specified text, icon, and selection state.

Class Methods

Sr.No.	Method & Description
1	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JCheckBoxMenuItem.
2	**Object[] getSelectedObjects()** Returns an array (length 1) containing the checkbox menu item label or null if the checkbox is not selected.
3	**boolean getState()**

	Returns the selected-state of the item.
4	**String getUIClassID()** Returns the name of the L&F class that renders this component.
5	**protected String paramString()** Returns a string representation of this JCheckBoxMenuItem.
6	**void setState(boolean b)** Sets the selected-state of the item.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.JMenuItem
- javax.swing.JAbstractButton
- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JCheckboxMenuItem Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingMenuDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingMenuDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingMenuDemo(){
      prepareGUI();
   }
```

```
public static void main(String[] args){
   SwingMenuDemo  swingMenuDemo = new SwingMenuDemo();
   swingMenuDemo.showMenuDemo();
}

private void prepareGUI(){
   mainFrame = new JFrame("Java SWING Examples");
   mainFrame.setSize(400,400);
   mainFrame.setLayout(new GridLayout(3, 1));

   headerLabel = new JLabel("",JLabel.CENTER );
   statusLabel = new JLabel("",JLabel.CENTER);




   statusLabel.setSize(350,100);
   mainFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent windowEvent){
         System.exit(0);
      }
   });
   controlPanel = new JPanel();
   controlPanel.setLayout(new FlowLayout());

   mainFrame.add(headerLabel);
   mainFrame.add(controlPanel);
   mainFrame.add(statusLabel);
   mainFrame.setVisible(true);
}

private void showMenuDemo(){
   //create a menu bar
   final JMenuBar menuBar = new JMenuBar();

   //create menus
```

```
JMenu fileMenu = new JMenu("File");
JMenu editMenu = new JMenu("Edit");
final JMenu aboutMenu = new JMenu("About");
final JMenu linkMenu = new JMenu("Links");

//create menu items
JMenuItem newMenuItem = new JMenuItem("New");
newMenuItem.setMnemonic(KeyEvent.VK_N);
newMenuItem.setActionCommand("New");

JMenuItem openMenuItem = new JMenuItem("Open");
openMenuItem.setActionCommand("Open");

JMenuItem saveMenuItem = new JMenuItem("Save");
saveMenuItem.setActionCommand("Save");

JMenuItem exitMenuItem = new JMenuItem("Exit");
exitMenuItem.setActionCommand("Exit");

JMenuItem cutMenuItem = new JMenuItem("Cut");
cutMenuItem.setActionCommand("Cut");

JMenuItem copyMenuItem = new JMenuItem("Copy");
copyMenuItem.setActionCommand("Copy");

JMenuItem pasteMenuItem = new JMenuItem("Paste");
pasteMenuItem.setActionCommand("Paste");

MenuItemListener menuItemListener = new MenuItemListener();

newMenuItem.addActionListener(menuItemListener);
openMenuItem.addActionListener(menuItemListener);
saveMenuItem.addActionListener(menuItemListener);
exitMenuItem.addActionListener(menuItemListener);
cutMenuItem.addActionListener(menuItemListener);
copyMenuItem.addActionListener(menuItemListener);
pasteMenuItem.addActionListener(menuItemListener);
```

```
      final JCheckBoxMenuItem showWindowMenu =
         new JCheckBoxMenuItem("Show About", true);
      showWindowMenu.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
            if(showWindowMenu.getState()){
               menuBar.add(aboutMenu);
            }else{
               menuBar.remove(aboutMenu);
            }
         }
      });

      final JRadioButtonMenuItem showLinksMenu = new JRadioButtonMenuItem("Show
Links", true);
      showLinksMenu.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
            if(menuBar.getMenu(3)!= null){
               menuBar.remove(linkMenu);
               mainFrame.repaint();
            }else{
               menuBar.add(linkMenu);
               mainFrame.repaint();
            }
         }
      });

      //add menu items to menus
      fileMenu.add(newMenuItem);
      fileMenu.add(openMenuItem);
      fileMenu.add(saveMenuItem);
      fileMenu.addSeparator();
      fileMenu.add(showWindowMenu);
      fileMenu.addSeparator();
      fileMenu.add(showLinksMenu);
      fileMenu.addSeparator();
      fileMenu.add(exitMenuItem);
```

```
      editMenu.add(cutMenuItem);
      editMenu.add(copyMenuItem);
      editMenu.add(pasteMenuItem);

      //add menu to menubar
      menuBar.add(fileMenu);
      menuBar.add(editMenu);
      menuBar.add(aboutMenu);
      menuBar.add(linkMenu);

      //add menubar to the frame
      mainFrame.setJMenuBar(menuBar);
      mainFrame.setVisible(true);
   }

   class MenuItemListener implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         statusLabel.setText(e.getActionCommand()
         + " JMenuItem clicked.");
      }
   }
}
```

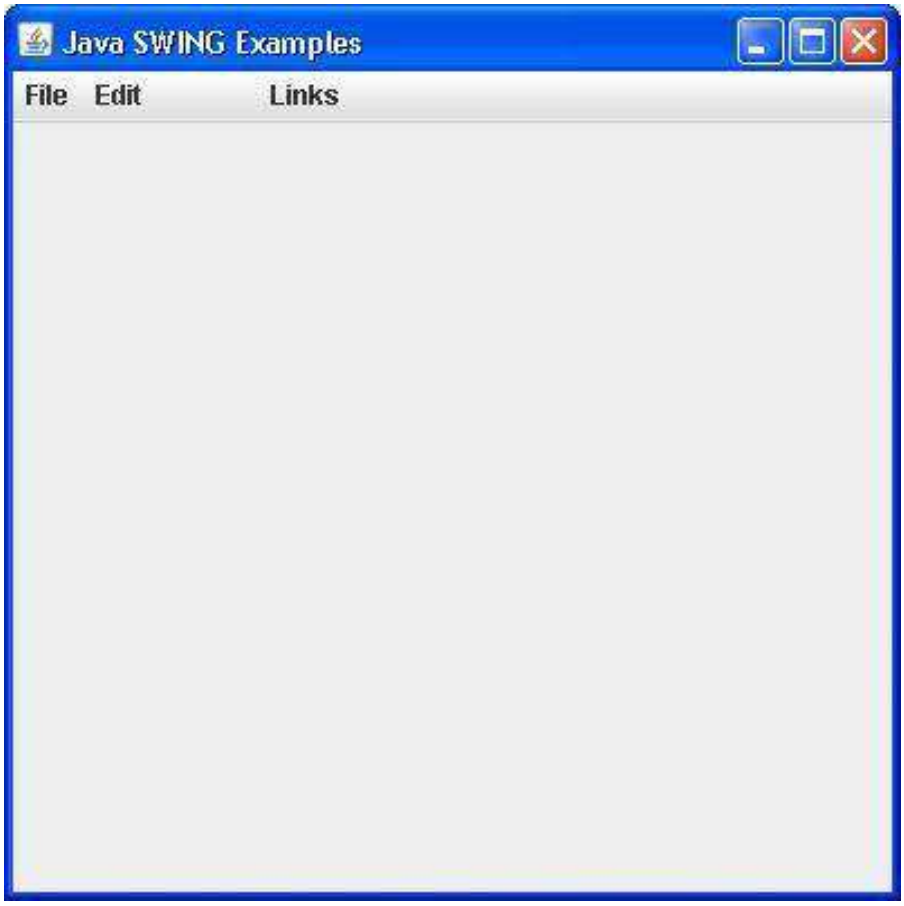
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingMenuDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingMenuDemo
```

Verify the following output. (Click on File Menu. Unselect "Show About" menu item.)



JRadioButtonMenuItem Class

Introduction

The JRadioButtonMenuItem class represents a checkbox which can be included in a menu. Selecting the checkbox in the menu changes the control's state from **on** to **off** or from **off** to **on**.

Class Declaration

Following is the declaration for **javax.swing.JRadioButtonMenuItem** class:

```
public class JRadioButtonMenuItem
   extends JMenuItem
      implements Accessible
```

Class Constructors

Sr.No.	Constructor & Description
1	**JRadioButtonMenuItem()** Creates a JRadioButtonMenuItem with no set text or icon.
2	**JRadioButtonMenuItem(Action a)** Creates a radio button menu item whose properties are taken from the Action supplied.
3	**JRadioButtonMenuItem(Icon icon)** Creates a JRadioButtonMenuItem with an icon.
4	**JRadioButtonMenuItem(Icon icon, boolean selected)** Creates a radio button menu item with the specified image and selection state, but no text.
5	**JRadioButtonMenuItem(String text)** Creates a JRadioButtonMenuItem with text.
6	**JRadioButtonMenuItem(String text, boolean selected)** Creates a radio button menu item with the specified text and selection state.
7	**JRadioButtonMenuItem(String text, Icon icon)** Creates a radio button menu item with the specified text and Icon.
8	**JRadioButtonMenuItem(String text, Icon icon, boolean selected)** Creates a radio button menu item with the specified text, image, and selection state.

Class Methods

Sr.No.	Method & Description
1	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JRadioButtonMenuItem.
2	**String getUIClassID()** Returns the name of the L&F class that renders this component.
3	**protected String paramString()** Returns a string representation of this JRadioButtonMenuItem.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.JMenuItem
- javax.swing.JAbstractButton
- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JRadioButtonMenuItem Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingMenuDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingMenuDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;
```

```
   public SwingMenuDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingMenuDemo  swingMenuDemo = new SwingMenuDemo();
      swingMenuDemo.showMenuDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java SWING Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showMenuDemo(){
      //create a menu bar
      final JMenuBar menuBar = new JMenuBar();

      //create menus
```

```
JMenu fileMenu = new JMenu("File");
JMenu editMenu = new JMenu("Edit");
final JMenu aboutMenu = new JMenu("About");
final JMenu linkMenu = new JMenu("Links");

//create menu items
JMenuItem newMenuItem = new JMenuItem("New");
newMenuItem.setMnemonic(KeyEvent.VK_N);
newMenuItem.setActionCommand("New");

JMenuItem openMenuItem = new JMenuItem("Open");
openMenuItem.setActionCommand("Open");

JMenuItem saveMenuItem = new JMenuItem("Save");
saveMenuItem.setActionCommand("Save");

JMenuItem exitMenuItem = new JMenuItem("Exit");
exitMenuItem.setActionCommand("Exit");

JMenuItem cutMenuItem = new JMenuItem("Cut");
cutMenuItem.setActionCommand("Cut");

JMenuItem copyMenuItem = new JMenuItem("Copy");
copyMenuItem.setActionCommand("Copy");

JMenuItem pasteMenuItem = new JMenuItem("Paste");
pasteMenuItem.setActionCommand("Paste");

MenuItemListener menuItemListener = new MenuItemListener();

newMenuItem.addActionListener(menuItemListener);
openMenuItem.addActionListener(menuItemListener);
saveMenuItem.addActionListener(menuItemListener);
exitMenuItem.addActionListener(menuItemListener);
cutMenuItem.addActionListener(menuItemListener);
copyMenuItem.addActionListener(menuItemListener);
pasteMenuItem.addActionListener(menuItemListener);
```

```
      final JCheckBoxMenuItem showWindowMenu = new JCheckBoxMenuItem("Show
About", true);
      showWindowMenu.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
            if(showWindowMenu.getState()){
               menuBar.add(aboutMenu);
            }else{
               menuBar.remove(aboutMenu);
            }
         }
      });

      final JRadioButtonMenuItem showLinksMenu =
         new JRadioButtonMenuItem("Show Links", true);
      showLinksMenu.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
            if(menuBar.getMenu(3)!= null){
               menuBar.remove(linkMenu);
               mainFrame.repaint();
            }else{
               menuBar.add(linkMenu);
               mainFrame.repaint();
            }
         }
      });


      //add menu items to menus
      fileMenu.add(newMenuItem);
      fileMenu.add(openMenuItem);
      fileMenu.add(saveMenuItem);
      fileMenu.addSeparator();
      fileMenu.add(showWindowMenu);
      fileMenu.addSeparator();
      fileMenu.add(showLinksMenu);
      fileMenu.addSeparator();
```

```
      fileMenu.add(exitMenuItem);
      editMenu.add(cutMenuItem);
      editMenu.add(copyMenuItem);
      editMenu.add(pasteMenuItem);

      //add menu to menubar
      menuBar.add(fileMenu);
      menuBar.add(editMenu);
      menuBar.add(aboutMenu);
      menuBar.add(linkMenu);

      //add menubar to the frame
      mainFrame.setJMenuBar(menuBar);
      mainFrame.setVisible(true);
   }

   class MenuItemListener implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         statusLabel.setText(e.getActionCommand()
         + " JMenuItem clicked.");
      }
   }
}
```

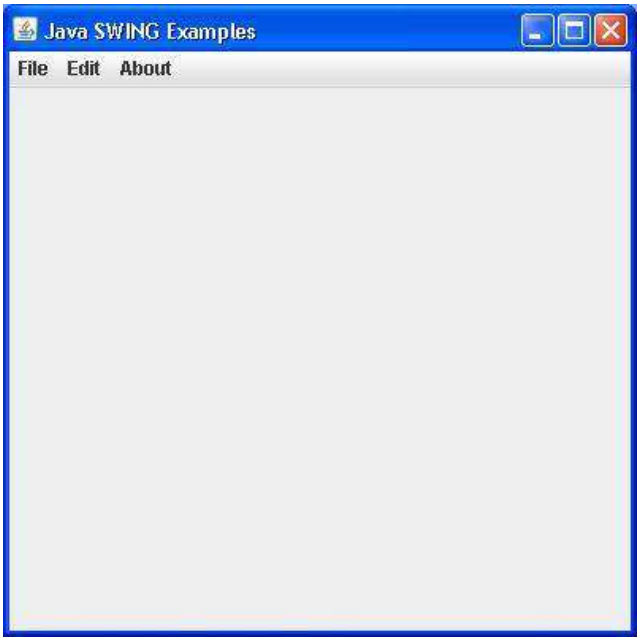
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingMenuDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingMenuDemo
```

Verify the following output. (Click on File Menu. Unselect "Show Link" menu item.)



JPopupMenu Class

Introduction

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

Class Declaration

Following is the declaration for **javax.swing.JPopupMenu** class:

```
public class JPopupMenu
   extends JComponent
      implements Accessible, MenuElement
```

Class Constructors

Sr.No.	Constructor & Description
1	**JPopupMenu()** Constructs a JPopupMenu without an "invoker".
2	**JPopupMenu(String label)** Constructs a JPopupMenu with the specified title.

Class Methods

Sr.No.	Method & Description
1	**JMenuItem add(Action a)** Appends a new menu item to the end of the menu which dispatches the specified Action object.
2	**JMenuItem add(JMenuItem menuItem)** Appends the specified menu item to the end of this menu.
3	**JMenuItem add(String s)** Creates a new menu item with the specified text and appends it to the end of this menu.
4	**void addMenuKeyListener(MenuKeyListener l)** Adds a MenuKeyListener to the popup menu.
5	**void addPopupMenuListener(PopupMenuListener l)** Adds a PopupMenu listener.
6	**void addSeparator()** Appends a new separator at the end of the menu.
7	**protected PropertyChangeListener createActionChangeListener(JMenuItem b)** Returns a properly configured PropertyChangeListener which updates the control as changes to the Action occur.
8	**protected JMenuItem createActionComponent(Action a)** Factory method which creates the JMenuItem for Actions added to the JPopupMenu.
9	**protected void firePopupMenuCanceled()** Notifies PopupMenuListeners that this popup menu is cancelled.
10	**protected void firePopupMenuWillBecomeInvisible()**

	Notifies PopupMenuListeners that this popup menu will become invisible.
11	**protected void firePopupMenuWillBecomeVisible()** Notifies PopupMenuListeners that this popup menu will become visible.
12	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JPopupMenu.
13	**Component getComponent()** Returns this JPopupMenu component.
14	**Component getComponentAtIndex(int i)** Deprecated. Replaced by Container.getComponent(int)
15	**int getComponentIndex(Component c)** Returns the index of the specified component.
16	**static boolean getDefaultLightWeightPopupEnabled()** Gets the defaultLightWeightPopupEnabled property, which by default is true.
17	**Component getInvoker()** Returns the component which is the 'invoker' of this popup menu.
18	**String getLabel()** Returns the popup menu's label.
19	**Insets getMargin()** Returns the margin, in pixels, between the popup menu's border and its containees.
20	**MenuKeyListener[]getMenuKeyListeners()** Returns an array of all the MenuKeyListeners added to this JPopupMenu with addMenuKeyListener().
21	**PopupMenuListener[] getPopupMenuListeners()** Returns an array of all the PopupMenuListeners added to this JMenuItem with addPopupMenuListener().

22	**SingleSelectionModel getSelectionModel()** Returns the model object that handles single selections.
23	**MenuElement[] getSubElements()** Returns an array of MenuElements containing the submenu for this menu component.
24	**PopupMenuUI getUI()** Returns the look and feel (L&F) object that renders this component.
25	**String getUIClassID()** Returns the name of the L&F class that renders this component.
26	**void insert(Action a, int index)** Inserts a menu item for the specified Action object at a given position.
27	**void insert(Component component, int index)** Inserts the specified component into the menu at a given position.
28	**boolean isBorderPainted()** Checks whether the border should be painted.
29	**boolean isLightWeightPopupEnabled()** Gets the lightWeightPopupEnabled property.
30	**boolean isPopupTrigger(MouseEvent e)** Returns true if the MouseEvent is considered a popup trigger by the JPopupMenu's currently installed UI.
31	**boolean isVisible()** Returns true if the popup menu is visible (currently being displayed).
32	**void menuSelectionChanged(boolean isIncluded)** Messaged when the menubar selection changes to activate or deactivate this menu.
33	**void pack()**

	Lays out the container so that it uses the minimum space needed to display its contents.
34	**protected void paintBorder(Graphics g)** Paints the popup menu's border if the borderPainted property is true.
35	**protected String paramString()** Returns a string representation of this JPopupMenu.
36	**protected void processFocusEvent(FocusEvent evt)** Processes focus events occurring on this component by dispatching them to any registered FocusListener objects.
37	**protected void processKeyEvent(KeyEvent evt)** Processes key stroke events such as mnemonics and accelerators.
38	**void processKeyEvent(KeyEvent e, MenuElement[] path, MenuSelectionManager manager)** Processes a key event forwarded from the MenuSelectionManager and changes the menu selection, if necessary, by using MenuSelectionManager's API.
39	**void processMouseEvent(MouseEvent event, MenuElement[] path, MenuSelectionManager manager)** This method is required to conform to the MenuElement interface, but it not implemented.
40	**void remove(int pos)** Removes the component at the specified index from this popup menu.
41	**void removeMenuKeyListener(MenuKeyListener l)** Removes a MenuKeyListener from the popup menu.
42	**void removePopupMenuListener(PopupMenuListener l)** Removes a PopupMenu listener.
43	**void setBorderPainted(boolean b)** Sets whether the border should be painted.
44	

	static void setDefaultLightWeightPopupEnabled(boolean aFlag) Sets the default value of the lightWeightPopupEnabled property.
45	**void setInvoker(Component invoker)** Sets the invoker of this popup menu - the component in which the popup menu menu is to be displayed.
46	**void setLabel(String label)** Sets the popup menu's label.
47	**void setLightWeightPopupEnabled(boolean aFlag)** Sets the value of the lightWeightPopupEnabled property, which by default is true.
48	**void setLocation(int x, int y)** Sets the location of the upper left corner of the popup menu using **x, y** coordinates.
49	**void setPopupSize(Dimension d)** Sets the size of the Popup window using a Dimension object.
50	**void setPopupSize(int width, int height)** Sets the size of the Popup window to the specified width and height.
51	**void setSelected(Component sel)** Sets the currently selected component, This will result in a change in the selection model.
52	**void setSelectionModel(SingleSelectionModel model)** Sets the model object to handle single selections.
53	**void setUI(PopupMenuUI ui)** Sets the L&F object that renders this component.
54	**void setVisible(boolean b)** Sets the visibility of the popup menu.

55	**void show(Component invoker, int x, int y)** Displays the popup menu at the position **x,y** in the coordinate space of the component invoker.
56	**void updateUI()** Resets the UI property to a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JPopupMenu Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingMenuDemo.java

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

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;

public class SwingJpopupMenu{
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingJpopupMenu(){
      prepareGUI();
   }
```

```
public static void main(String[] args){
SwingJpopupMenu  swingMenuDemo = new SwingJpopupMenu();
   swingMenuDemo.showPopupMenuDemo();
}

private void prepareGUI(){
   mainFrame = new JFrame("Java SWING Examples");
   mainFrame.setSize(400,400);
   mainFrame.setLayout(new GridLayout(3, 1));

   headerLabel = new JLabel("",JLabel.CENTER );
   statusLabel = new JLabel("",JLabel.CENTER);

   statusLabel.setSize(350,100);
   mainFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent windowEvent){
         System.exit(0);
      }
   });
   controlPanel = new JPanel();
   controlPanel.setLayout(new FlowLayout());

   mainFrame.add(headerLabel);
   mainFrame.add(controlPanel);
   mainFrame.add(statusLabel);
   mainFrame.setVisible(true);
}

private void showPopupMenuDemo(){
   final JPopupMenu editMenu = new JPopupMenu("Edit");

   JMenuItem cutMenuItem = new JMenuItem("Cut");
   cutMenuItem.setActionCommand("Cut");

   JMenuItem copyMenuItem = new JMenuItem("Copy");
   copyMenuItem.setActionCommand("Copy");

   JMenuItem pasteMenuItem = new JMenuItem("Paste");
   pasteMenuItem.setActionCommand("Paste");
```

```
      MenuItemListener menuItemListener = new MenuItemListener();

      cutMenuItem.addActionListener(menuItemListener);
      copyMenuItem.addActionListener(menuItemListener);
      pasteMenuItem.addActionListener(menuItemListener);

      editMenu.add(cutMenuItem);
      editMenu.add(copyMenuItem);
      editMenu.add(pasteMenuItem);

      mainFrame.addMouseListener(new MouseAdapter() {
         public void mouseClicked(MouseEvent e) {
            editMenu.show(mainFrame, e.getX(), e.getY());
         }
      });
      mainFrame.add(editMenu);
      mainFrame.setVisible(true);
   }

   class MenuItemListener implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         statusLabel.setText(e.getActionCommand()
            + " MenuItem clicked.");
      }
   }
}
```

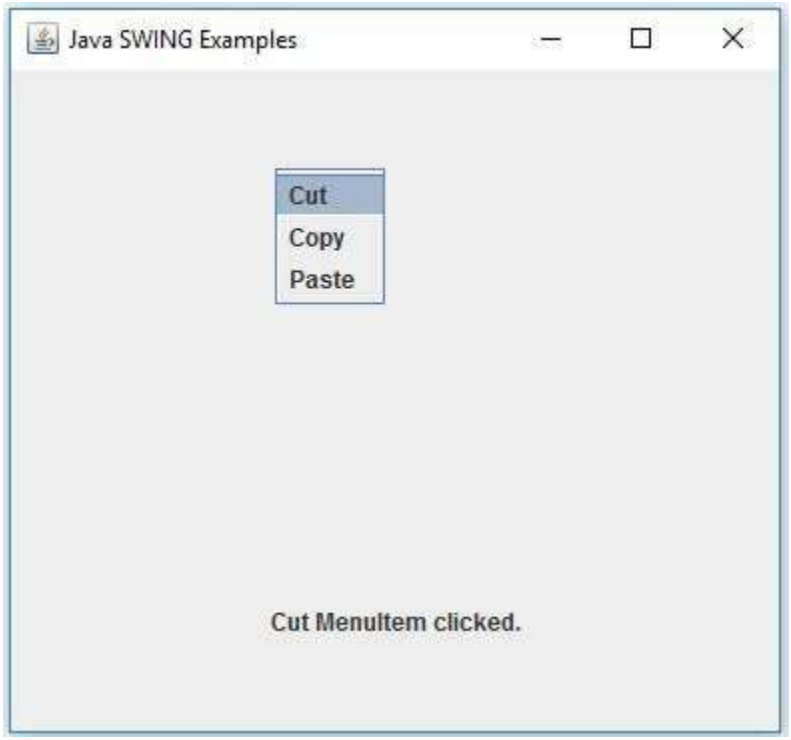
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingMenuDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingMenuDemo
```

Verify the following output. (Click in the middle on the screen.)



10. Swing – Containers

Containers are an integral part of SWING GUI components. A container provides a space where a component can be located. A Container in AWT is a component itself and it provides the capability to add a component to itself. Following are certain noticable points to be considered.

- Sub classes of Container are called as Container. For example, JPanel, JFrame and JWindow.
- Container can add only a Component to itself.
- A default layout is present in each container which can be overridden using **setLayout** method.

SWING Containers

Following is the list of commonly used containers while designed GUI using SWING.

Sr. No.	Container & Description
1	**Panel** JPanel is the simplest container. It provides space in which any other component can be placed, including other panels.
2	**Frame** A JFrame is a top-level window with a title and a border.
3	**Window** A JWindow object is a top-level window with no borders and no menubar.

JPanel Class

Introduction

The class **JPanel** is a generic lightweight container.

Class Declaration

Following is the declaration for **javax.swing.JPanel** class:

```
public class JPanel
   extends JComponent
      implements Accessible
```

Class Constructors

Sr.No.	Constructor & Description
1	**JPanel()** Creates a new JPanel with a double buffer and a flow layout.
2	**JPanel(boolean isDoubleBuffered)** Creates a new JPanel with FlowLayout and the specified buffering strategy.
3	**JPanel(LayoutManager layout)** Creates a new buffered JPanel with the specified layout manager.
4	**JPanel(LayoutManager layout, boolean isDoubleBuffered)** Creates a new JPanel with the specified layout manager and buffering strategy.

Class Methods

Sr.No.	Method & Description
1	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JPanel.

2	**PanelUI getUI()** Returns the look and feel (L&F) object that renders this component.
3	**String getUIClassID()** Returns a string that specifies the name of the L&F class which renders this component.
4	**protected String paramString()** Returns a string representation of this JPanel.
5	**void setUI(PanelUI ui)** Sets the look and feel (L&F) object that renders this component.
6	**void updateUI()** Resets the UI property with a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes:

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

JPanel Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingContainerDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingContainerDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
```

```
   private JLabel statusLabel;
   private JPanel controlPanel;
   private JLabel msglabel;

   public SwingContainerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingContainerDemo  swingContainerDemo = new SwingContainerDemo();
      swingContainerDemo.showJPanelDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      msglabel = new JLabel("Welcome to TutorialsPoint SWING Tutorial.",
JLabel.CENTER);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
```

```
   }

   private void showJPanelDemo(){
      headerLabel.setText("Container in action: JPanel");

      JPanel panel = new JPanel();
      panel.setBackground(Color.magenta);
      panel.setLayout(new FlowLayout());
      panel.add(msglabel);

      controlPanel.add(panel);
      mainFrame.setVisible(true);
   }
}
```

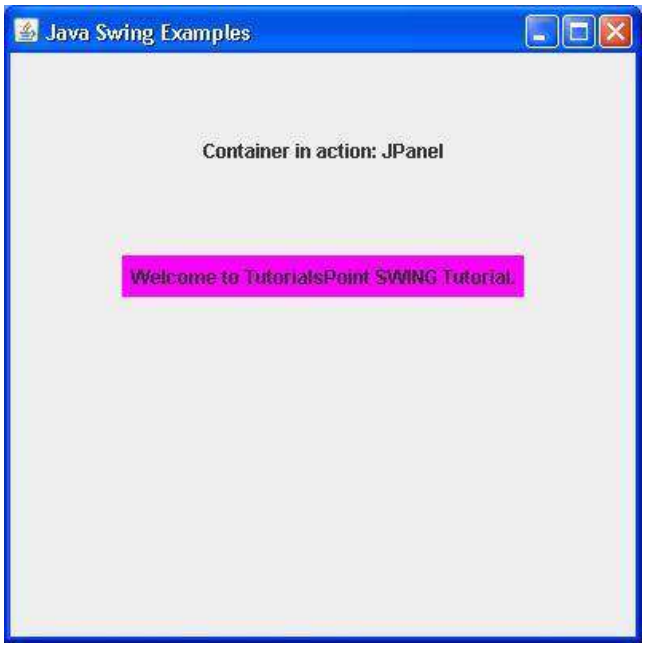
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingContainerDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingContainerDemo
```

Verify the following output.



JFrame Class

Introduction

The class **JFrame** is an extended version of **java.awt.Frame** that adds support for the JFC/Swing component architecture.

Class Declaration

Following is the declaration for **javax.swing.JFrame** class:

```
public class JFrame
   extends Frame
      implements WindowConstants, Accessible, RootPaneContainer
```

Field

Following are the fields for **java.awt.Component** class:

- **protected AccessibleContext accessibleContext** - The accessible context property.
- **static int EXIT_ON_CLOSE** - The exit application default window close operation.
- **protected JRootPane rootPane** - The JRootPane instance that manages the contentPane and optional menuBar for this frame, as well as the glassPane.
- **protected boolean rootPaneCheckingEnabled** - If true then calls to add and setLayout will be forwarded to the contentPane.

Class Constructors

Sr.No.	Constructor & Description
1	**JFrame()** Constructs a new frame that is initially invisible.
2	**JFrame(GraphicsConfiguration gc)** Creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title.
3	**JFrame(String title)** Creates a new, initially invisible Frame with the specified title.

4	**JFrame(String title, GraphicsConfiguration gc)** Creates a JFrame with the specified title and the specified GraphicsConfiguration of a screen device.

Class Methods

Sr.No.	Method & Description
1	**protected void addImpl(Component comp, Object constraints, int index)** Adds the specified child Component.
2	**protected JRootPane createRootPane()** Called by the constructor methods to create the default rootPane.
3	**protected void frameInit()** Called by the constructors to init the JFrame properly.
4	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JFrame.
5	**Container getContentPane()** Returns the contentPane object for this frame.
6	**int getDefaultCloseOperation()** Returns the operation that occurs when the user initiates a "close" on this frame.
7	**Component getGlassPane()** Returns the glassPane object for this frame.
8	**Graphics getGraphics()** Creates a graphics context for this component.
9	**JMenuBar getJMenuBar()**

	Returns the menubar set on this frame.
10	**JLayeredPane getLayeredPane()** Returns the layeredPane object for this frame.
11	**JRootPane getRootPane()** Returns the rootPane object for this frame.
12	**TransferHandler getTransferHandler()** Gets the transferHandler property.
13	**static boolean isDefaultLookAndFeelDecorated()** Returns true if the newly created JFrames have their Window decorations provided by the current look and feel.
14	**protected boolean isRootPaneCheckingEnabled()** Returns whether calls to add and setLayout are forwarded to the contentPane.
15	**protected String paramString()** Returns a string representation of this JFrame.
16	**protected void processWindowEvent(WindowEvent e)** Processes window events occurring on this component.
17	**void remove(Component comp)** Removes the specified component from the container.
18	**void repaint(long time, int x, int y, int width, int height)** Repaints the specified rectangle of this component within time milliseconds.
19	**void setContentPane(Container contentPane)** Sets the contentPane property.
20	**void setDefaultCloseOperation(int operation)** Sets the operation that will happen by default when the user initiates a "close" on this frame.
21	**static void setDefaultLookAndFeelDecorated(boolean defaultLookAndFeelDecorated)**

	Provides a hint as to whether or not newly created JFrames should have their Window decorations (such as borders, widgets to close the window, title...) provided by the current look and feel.
22	**void setGlassPane(Component glassPane)** Sets the glassPane property.
23	**void setIconImage(Image image)** Sets the image to be displayed as the icon for this window.
24	**void setJMenuBar(JMenuBar menubar)** Sets the menubar for this frame.
25	**void setLayeredPane(JLayeredPane layeredPane)** Sets the layeredPane property.
26	**void setLayout(LayoutManager manager)** Sets the LayoutManager.
27	**protected void setRootPane(JRootPane root)** Sets the rootPane property.
28	**protected void setRootPaneCheckingEnabled(boolean enabled)** Sets whether calls to add and setLayout are forwarded to the contentPane.
29	**void setTransferHandler(TransferHandler newHandler)** Sets the transferHandler property, which is a mechanism to support the transfer of data into this component.
30	**void update(Graphics g)** Just calls paint(g).

Methods Inherited

This class inherits methods from the following classes:

- java.awt.Frame
- java.awt.Window
- java.awt.Container

- java.awt.Component
- java.lang.Object

JFrame Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingContainerDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingContainerDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;
   private JLabel msglabel;

   public SwingContainerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingContainerDemo  swingContainerDemo = new SwingContainerDemo();
      swingContainerDemo.showJFrameDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
```

```
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      msglabel = new JLabel("Welcome to TutorialsPoint SWING Tutorial.",
JLabel.CENTER);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showJFrameDemo(){
      headerLabel.setText("Container in action: JFrame");

      final JFrame frame = new JFrame();
      frame.setSize(300, 300);
      frame.setLayout(new FlowLayout());
      frame.add(msglabel);
      frame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            frame.dispose();
         }
      });
      JButton okButton = new JButton("Open a Frame");
      okButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            statusLabel.setText("A Frame shown to the user.");
            frame.setVisible(true);
         }
      });
```

```
      controlPanel.add(okButton);
      mainFrame.setVisible(true);
   }
}
```

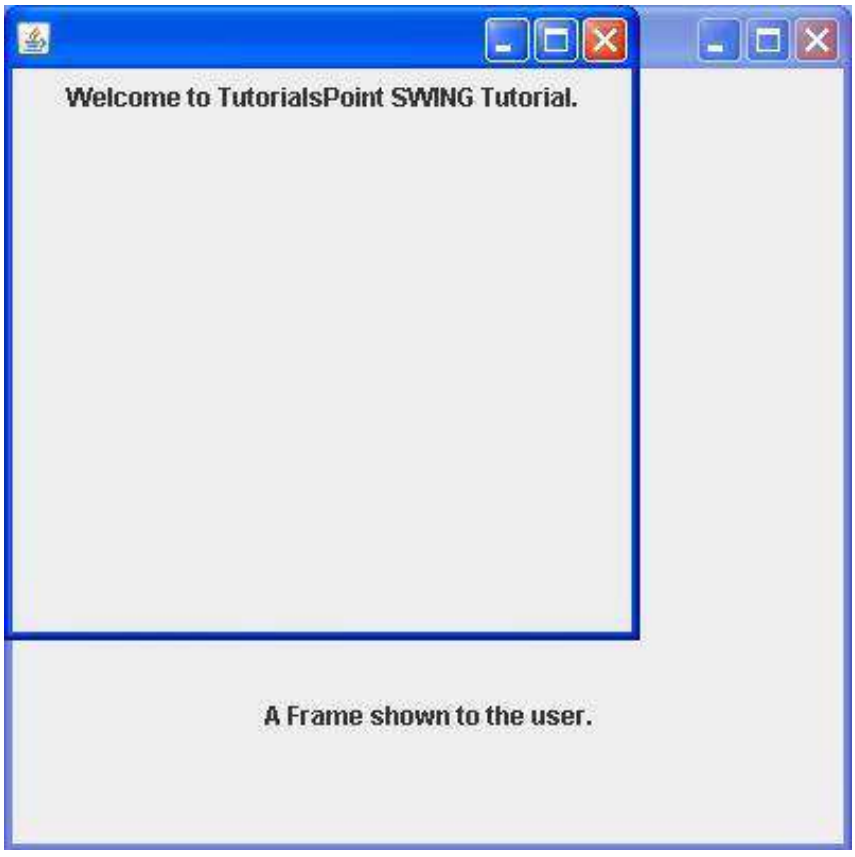
Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingContainerDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingContainerDemo
```

Verify the following output.



JWindow Class

Introduction

The class **JWindow** is a container that can be displayed but does not have the title bar or window-management buttons.

Class Declaration

Following is the declaration for **javax.swing.JWindow** class:

```
public class JWindow
   extends Window
      implements Accessible, RootPaneContainer
```

Field

Following are the fields for **java.awt.Component** class:

- **protected AccessibleContext accessibleContext** - The accessible context property.
- **protected JRootPane rootPane** - The JRootPane instance that manages the contentPane and optional menuBar for this frame, as well as the glassPane.
- **protected boolean rootPaneCheckingEnabled** - If true then calls to add and setLayout will be forwarded to the contentPane.

Class Constructors

Sr.No.	Constructor & Description
1	**JWindow()** Creates a window with no specified owner.
2	**JWindow(Frame owner)** Creates a window with the specified owner frame.
3	**JWindow(GraphicsConfiguration gc)** Creates a window with the specified GraphicsConfiguration of a screen device.
4	**JWindow(Window owner)** Creates a window with the specified owner window.
5	**JWindow(Window owner, GraphicsConfiguration gc)** Creates a window with the specified owner window and GraphicsConfiguration of a screen device.

Class Methods

Sr.No.	Method & Description
1	**protected void addImpl(Component comp, Object constraints, int index)** Adds the specified child Component.
2	**protected JRootPane createRootPane()** Called by the constructor methods to create the default rootPane.
3	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JWindow.
4	**Container getContentPane()** Returns the Container which is the contentPane for this window.
5	**Component getGlassPane()** Returns the glassPane Component for this window.
6	**Graphics getGraphics()** Creates a graphics context for this component.
7	**JLayeredPane getLayeredPane()** Returns the layeredPane object for this window.
8	**JRootPane getRootPane()** Returns the rootPane object for this window.
9	**TransferHandler getTransferHandler()** Gets the transferHandler property.
10	**protected boolean isRootPaneCheckingEnabled()** Returns whether calls to add and setLayout are forwarded to the contentPane.
11	**protected String paramString()** Returns a string representation of this JWindow.
12	**void remove(Component comp)**

	Removes the specified component from the container.
13	**void repaint(long time, int x, int y, int width, int height)** Repaints the specified rectangle of this component within time milliseconds.
14	**void setContentPane(Container contentPane)** Sets the contentPane property for this window.
15	**void setGlassPane(Component glassPane)** Sets the glassPane property.
16	**void setLayeredPane(JLayeredPane layeredPane)** Sets the layeredPane property.
17	**void setLayout(LayoutManager manager)** Sets the LayoutManager.
18	**protected void setRootPane(JRootPane root)** Sets the new rootPane object for this window.
19	**protected void setRootPaneCheckingEnabled(boolean enabled)** Sets whether calls to add and setLayout are forwarded to the contentPane.
20	**void setTransferHandler(TransferHandler newHandler)** Sets the transferHandler property, which is a mechanism to support the transfer of data into this component.
21	**void update(Graphics g)** Calls paint(g).
22	**protected void windowInit()** Called by the constructors to init the JWindow properly.

Methods Inherited

This class inherits methods from the following classes:

- java.awt.Window
- java.awt.Container
- java.awt.Component
- java.lang.Object

JWindow Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingContainerDemo.java

```
package com.tutorialspoint.gui;

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

public class SwingContainerDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;
   private JLabel msglabel;

   public SwingContainerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingContainerDemo  swingContainerDemo = new SwingContainerDemo();
      swingContainerDemo.showJWindowDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
```

```
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);

         }

      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      msglabel = new JLabel("Welcome to TutorialsPoint SWING Tutorial."
         , JLabel.CENTER);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showJWindowDemo(){
      headerLabel.setText("Container in action: JWindow");
      final MessageWindow window = new MessageWindow(mainFrame, "Welcome to
TutorialsPoint SWING Tutorial.");

      JButton okButton = new JButton("Open a Window");
      okButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            window.setVisible(true);
            statusLabel.setText("A Window shown to the user.");
         }
      });
```

```
      controlPanel.add(okButton);
      mainFrame.setVisible(true);
   }

   class MessageWindow extends JWindow{
      private String message;

      public MessageWindow(JFrame parent, String
         message) {
         super(parent);
         this.message = message;
         setSize(300, 300);
         setLocationRelativeTo(parent);
      }

      public void paint(Graphics g)
      {
         super.paint(g);
         g.drawRect(0,0,getSize().width - 1,getSize().height - 1);
         g.drawString(message,50,150);
      }
   }
}
```

Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingContainerDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingContainerDemo
```

Verify the following output.