AWT

abstract window toolkit

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 (GUI) objects in an platform independent way. Abstract Window Toolkit (AWT) is a set of APIs used by Java programmers to create GUI objects. In this tutorial, we will learn how to use AWT to create GUI objects such as buttons, scroll bars, layout, menus, and more.

Audience

This tutorial is designed for all those software professionals who would like to learn JAVA GUI Programming in simple and easy steps.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of Java programming language and how to use it in practice.

Copyright & Disclaimer

Table of Contents

1. AWT – OVERVIEW

Graphical User Interface

Graphical User Interface (GUI) offers user interaction via some graphical components. For example, our underlying Operating System also offers GUI via window, frame, Panel, Button, Textfield, TextArea, Listbox, Combobox, Label, Checkbox etc. These all are known as components. Using these components, we can create an interactive user interface for an application.

GUI provides result to end-users in response to its raised events. It is entirely based on events. For example, clicking on a button, closing a window, opening a window, typing something in a text area etc. These activities are known as events. GUI makes it easier for the end user to use an application. It also makes them interesting.

Basic Terminologies

Term	Description
组件 Component	Component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. For example, buttons, checkboxes, list and scrollbars of a graphical user interface.
容器 Container	Container object is a component that can contain other components. Components added to a container are tracked in a list. The order of the list will define the components' front-to-back stacking order within the container. If no index is specified when adding a component to a container, it will be added to the end of the list.
面板 Panel	Panel provides space in which an application can attach any other components, including other panels.
Window	Window is a rectangular area, which is displayed on the screen. In a different window, we can execute different program and display different data. Window provide us with multitasking environment. A window must have either a frame, dialog, or another window defined as its owner when it's constructed.
框架 Frame	A Frame is a top-level window with a title and a border. The size of the frame includes any area designated for the border. Frame encapsulates **window**. It and has a title bar, menu bar, borders, and resizing corners.

帆布 Canvas	Canvas component represents a blank rectangular area of the screen onto which the application can draw. Application can also trap input events from the use of the blank area of Canvas component.

Examples of GUI Based Applications

Following are some of the examples for GUI based applications:

- Automated Teller Machine (ATM)
- Airline Ticketing System
- Information Kiosks at railway stations
- Mobile Applications
- Navigation Systems

Advantages of GUI over CUI

CUI字符用户界面

- GUI provides graphical icons to interact while the CUI (Character User Interface) offers the simple text-based interfaces.
- GUI makes the application more entertaining and interesting on the other hand CUI does not.
- GUI offers click and execute environment while in CUI every time, we have to enter the command for a task.
- New user can easily interact with graphical user interface by the visual indicators, but it is difficult in Character user interface.
- GUI offers a lot of controls of file system and the operating system while in CUI, you have to use commands, which is difficult to remember.
- Windows concept in GUI allows the user to view, manipulate, and control the multiple applications at once while in CUI, user can control one task at a time.
- GUI provides multitasking environment so as the CUI also does, but CUI does not provide same ease as the GUI do.
- Using GUI, it is easier to control and navigate the operating system, which becomes very slow in command user interface.
- GUI can be easily customized but CUI cannot be.

2. AWT – ENVIRONMENT SETUP

This chapter guides you on how to download and set up Java on your computer. Please follow the steps given below to set up the environment.

Java SE is freely available on the link Download Java. So you download a version based on your operating system.

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

Setting up the Path for Windows 2000/XP

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

- Right-click on 'My Computer' and select 'Properties'.
- Click on the 'Environment variables' button under the 'Advanced' tab.
- Now alter the 'Path' variable so that it also contains the path to the Java executable. For 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:

- 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 — 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 of your '.bashrc: export PATH=/path/to/java:$PATH'

Popular Java Editors

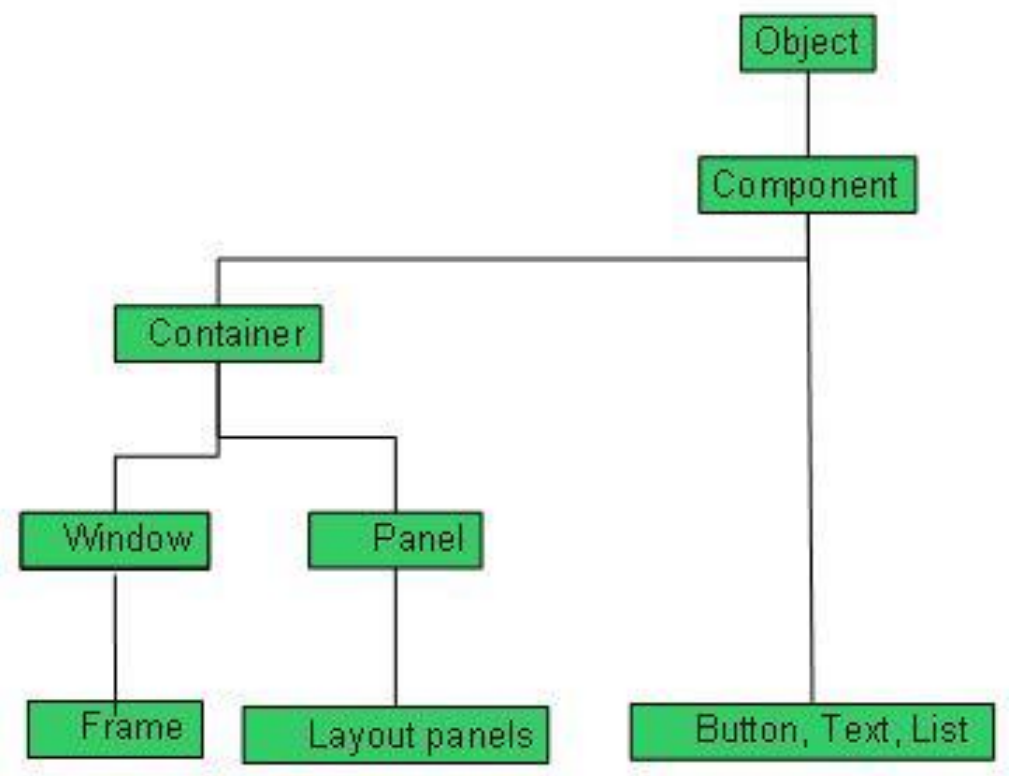
To write your java programs, you will need a text editor. There are even more sophisticated IDE available in the market. But for now, you can consider one of the following:

- **Notepad:** On Windows system, you can use any simple text editor like Notepad (Recommended for this tutorial), TextPad.
- **Netbeans:** It is a Java IDE that is open source and free, it can be downloaded from http://www.netbeans.org/index.html.
- **Eclipse:** It is also a java IDE developed by the eclipse open source community and can be downloaded from http://www.eclipse.org/.

3. AWT – 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. We will discuss all these 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 Layout chapter.
- **Behavior:** These are events that occur when the user interacts with UI elements. This part will be covered in Event Handling chapter.



Every AWT controls inherit properties from Component class.

Sr. No.	Control & Description
1	**Component** A Component is an abstract super class for GUI controls and it represents an object with graphical representation.

AWT Component Class

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

alignment对准

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_ALIGNMENT** -- Ease-of-use constant for getAlignmentX.
- **static float TOP_ALIGNMENT** -- Ease-of-use constant for getAlignmentY().

Class Constructors

S.N.	Constructor & Description

1	**protected Component()** This creates a new Component.

Class Methods

S.N.	Method & Description
1	boolean action(Event evt, Object what) Deprecated. As of JDK version 1.1, should register this component as ActionListener on 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 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 addProperty ChangeListener(String propertyName, Property ChangeListener 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 points 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)** Support for 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)** Support for 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 return 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 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 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 return 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	**Property ChangeListener[] getProperty ChangeListeners (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 return 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 it native screen resource.
170	**void removePropertyChangeListener(PropertyChangeListener listener)** Removes a PropertyChangeListener from the listener list.
171	**void remove Property ChangeListener(String propertyName, PropertyChange Listener 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)** Associate 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)** Support for 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 classes:

- java.lang.Object

AWT UI Elements

Following is the list of commonly used controls while designing GUI using AWT:

Sr. No.	Control & Description
1	**Label** A Label object is a component for placing text in a container.
2	**Button** This class creates a labeled button.
3	**Check Box** A check box is a graphical component that can be in either an **on** (true) or **off** (false) state.
4	**Check Box Group** The CheckboxGroup class is used to group the set of checkbox.
5	**List** The List component presents the user with a scrolling list of text items.
6	**Text Field** A TextField object is a text component that allows for the editing of a single line of text.
7	**Text Area** A TextArea object is a text component that allows for the editing of a multiple lines of text.

8	**Choice** A Choice control is used to show pop up menu of choices. Selected choice is shown on the top of the menu.
9	**Canvas** A Canvas control represents a rectangular area where application can draw something or can receive inputs created by user.
10	**Image** An Image control is superclass for all image classes representing graphical images.
11	**Scroll Bar** A Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
12	**Dialog** A Dialog control represents a top-level window with a title and a border used to take some form of input from the user.
13	**File Dialog** A FileDialog control represents a dialog window from which the user can select a file.

AWT Label Class

Label is a passive control because it does not create any event when accessed by the user. The label control is an object of Label. A label displays a single line of read-only text. However, the text can be changed by the application programmer, but cannot be changed by the end user in any way.

Class Declaration

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

```
public class Label
   extends Component
      implements Accessible
```

Field

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

static int CENTER -- Indicates that the label should be centered.

static int LEFT -- Indicates that the label should be left justified.

static int RIGHT -- Indicates that the label should be right justified.

Class Constructors

S.N.	Constructor & Description
1	**Label()** Constructs an empty label.
2	**Label(String text)** Constructs a new label with the specified string of text, left justified.
3	**Label(String text, int alignment)** Constructs a new label that presents the specified string of text with the specified alignment.

Class Methods

S.N.	Method & Description
1	**void addNotify()** Creates the peer for this label.
2	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Label.
3	**int getAlignment()** Gets the current alignment of this label.
4	**String getText()** Gets the text of this label.
5	**protected String paramString()** Returns a string representing the state of this Label.
6	**void setAlignment(int alignment)** Sets the alignment for this label to the specified alignment.
7	**void setText(String text)** Sets the text for this label to the specified text.

Methods Inherited

This class inherits methods from the following classes:

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

Label Example

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

AwtControlDemo.java

```
package com.tutorialspoint.gui;


import java.awt.*;
```

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

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtControlDemo(){
      prepareGUI();
   }

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

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);
```

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

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

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

      Label label = new Label();
      label.setText("Welcome to TutorialsPoint AWT Tutorial.");
      label.setAlignment(Label.CENTER);
      label.setBackground(Color.GRAY);
      label.setForeground(Color.WHITE);
      controlPanel.add(label);

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

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

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

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

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

Output:

AWT Button Class

Button is a control component that has a label and generates an event when pressed. When a button is pressed and released, AWT sends an instance of ActionEvent to the button, by calling processEvent on the button. The button's processEvent method receives all events for the button; it passes an action event along by calling its own processActionEvent method. The latter method passes the action event on to any action listeners that have registered an interest in action events generated by this button.

If an application wants to perform some action based on a button being pressed and released, it should be implement ActionListener and register the new listener to receive events from this button, by calling the button's addActionListener method. The application can make use of the button's action command as a messaging protocol.

Class Declaration

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

```
public class Button
   extends Component
      implements Accessible
```

Class Constructors

S.N.	Constructor & Description
1	**Button()** Constructs a button with an empty string for its label.
2	**Button(String text)** Constructs a new button with specified label.

Class Methods

S.N.	Method & Description
1	**void addActionListener(ActionListener l)** Adds the specified action listener to receive action events from this button.
2	**void addNotify()** Creates the peer of the button.
3	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Button.
4	**String getActionCommand()** Returns the command name of the action event fired by this button.
5	**ActionListener[] getActionListeners()** Returns an array of all the action listeners registered on this button.
6	**String getLabel()** Gets the label of this button.
7	**<T extends EventListener> T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this Button.
8	**protected String paramString()** Returns a string representing the state of this Button.
9	**protected void processActionEvent(ActionEvent e)** Processes action events occurring on this button by dispatching them to any registered ActionListener objects.
10	**protected void processEvent(AWTEvent e)** Processes events on this button.
11	**void removeActionListener(ActionListener l)** Removes the specified action listener so that it no longer receives action events from this button.
12	**void setActionCommand(String command)** Sets the command name for the action event fired by this button.

13	**void setLabel(String label)** Sets the button's label to be the specified string.

Methods Inherited

This class inherits methods from the following classes:

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

Button Example

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

AwtControlDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtControlDemo  awtControlDemo = new AwtControlDemo();
      awtControlDemo.showButtonDemo();
   }
```

```
   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

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

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

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

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

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

```
            statusLabel.setText("Ok Button clicked.");
         }
      });

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

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

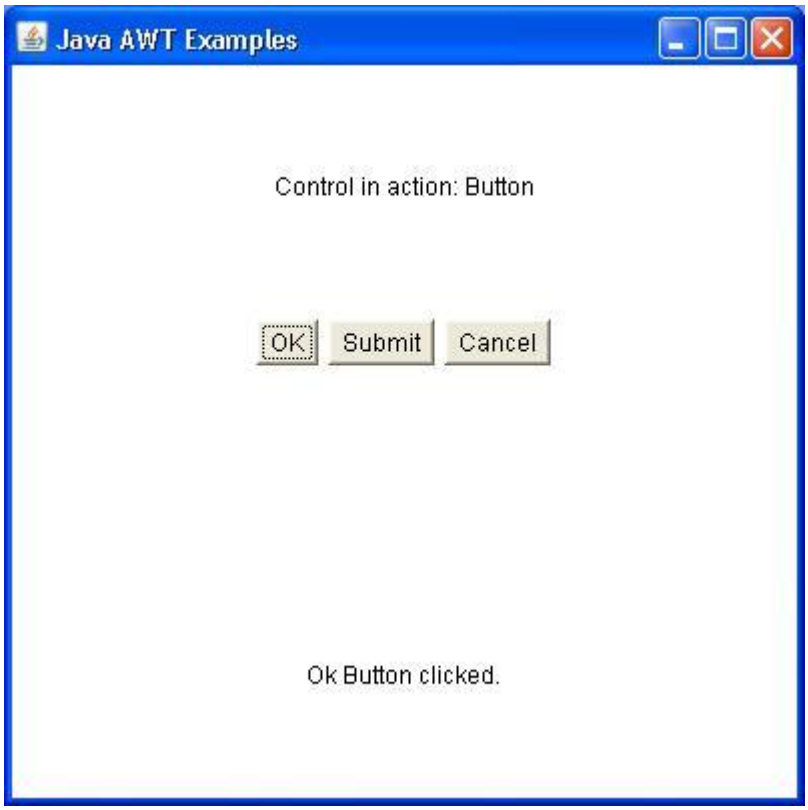
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output:



AWT CheckBox Class

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

Class Declaration

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

```
public class Checkbox
   extends Component
      implements ItemSelectable,Accessible
```

Class Constructors

S.N.	Constructor & Description
1	**Checkbox()** Creates a check box with an empty string for its label.
2	**Checkbox(String label)** Creates a check box with the specified label.
3	**Checkbox(String label, boolean state)**

	Creates a check box with the specified label and sets the specified state.
4	**Checkbox(String label, boolean state, CheckboxGroup group)** Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group.
5	**Checkbox(String label, CheckboxGroup group, boolean state)** Creates a check box with the specified label, in the specified check box group, and set to the specified state.

Class Methods

S.N.	Method & Description
1	**void addItemListener(ItemListener l)** Adds the specified item listener to receive item events from this check box.
2	**void addNotify()** Creates the peer of the Checkbox.
3	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Checkbox.
4	**CheckboxGroup getCheckboxGroup()** Determines this check box's group.
5	**ItemListener[] getItemListeners()** Returns an array of all the item listeners registered on this checkbox.
6	**String getLabel()** Gets the label of this check box.
7	**<T extends EventListener>T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this Checkbox.
8	**Object[] getSelectedObjects()** Returns an array (length 1) containing the checkbox label or null if the checkbox is not selected.
9	**boolean getState()** Determines whether this check box is in the **on** or **off** state.
10	**protected String paramString()** Returns a string representing the state of this Checkbox.
11	**protected void processEvent(AWTEvent e)** Processes events on this check box.
12	**protected void processItemEvent(ItemEvent e)** Processes item events occurring on this check box by dispatching them to any registered ItemListener objects.
13	**void removeItemListener(ItemListener l)** Removes the specified item listener so that the item listener no longer receives item events from this check box.
14	**void setCheckboxGroup(CheckboxGroup g)** Sets this check box's group to the specified check box group.
15	**void setLabel(String label)** Sets this check box's label to be the string argument.
16	**void setState(boolean state)** Sets the state of this check box to the specified state.

Methods Inherited

This class inherits methods from the following classes:

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

CheckBox Example

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

AwtControlDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtControlDemo(){
      prepareGUI();
   }

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

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      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");

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


      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);
   }
}
```

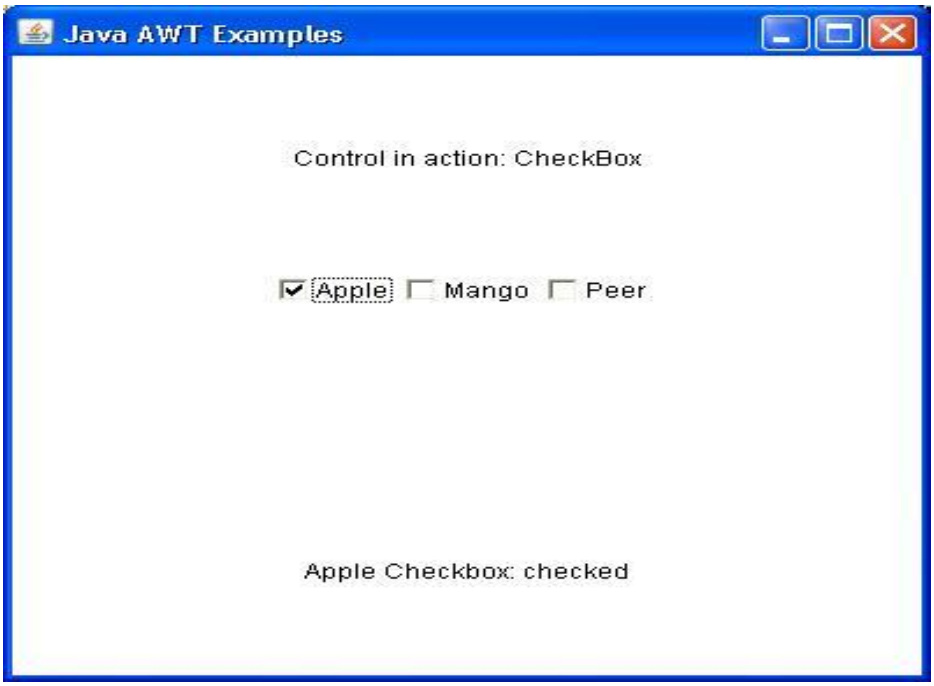
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output:



AWT CheckBoxGroup Class

The CheckboxGroup class is used to group the set of checkbox.

Class Declaration

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

```
public class CheckboxGroup
   extends Object
      implements Serializable
```

Class Constructors

S.N.	Constructor & Description
1	**CheckboxGroup() ()** Creates a new instance of CheckboxGroup.

Class Methods

S.N.	Method & Description
1	**Checkbox getCurrent()** Deprecated. As of JDK version 1.1, replaced by getSelectedCheckbox().
2	**Checkbox getSelectedCheckbox()** Gets the current choice from this check box group.
3	**void setCurrent(Checkbox box)**

	Deprecated. As of JDK version 1.1, replaced by setSelectedCheckbox(Checkbox).
4	**void setSelectedCheckbox(Checkbox box)** Sets the currently selected check box in this group to be the specified check box.
5	**String toString()** Returns a string representation of this check box group, including the value of its current selection.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

CheckBoxGroup Example

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

AwtControlDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtControlDemo  awtControlDemo = new AwtControlDemo();
```

```
      awtControlDemo.showCheckBoxGroupDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

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

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

   private void showCheckBoxGroupDemo(){

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

      CheckboxGroup fruitGroup = new CheckboxGroup();
```

```
      Checkbox chkApple = new Checkbox("Apple",fruitGroup,true);
      Checkbox chkMango = new Checkbox("Mango",fruitGroup,false);
      Checkbox chkPeer = new Checkbox("Peer",fruitGroup,false);

      statusLabel.setText("Apple Checkbox: checked");
      chkApple.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
            statusLabel.setText("Apple Checkbox: checked");
         }
      });

      chkMango.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
            statusLabel.setText("Mango Checkbox: checked");
         }
      });

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

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

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

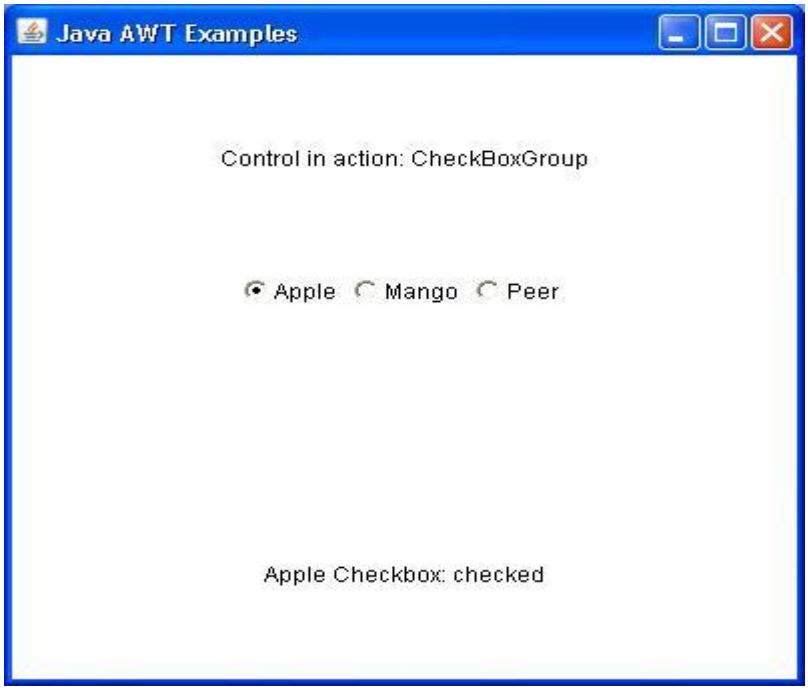
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output:



AWT List Class

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

Class Declaration

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

```
public class List
   extends Component
      implements ItemSelectable, Accessible
```

Class Constructors

S.N.	Constructor & Description
1	**List()** Creates a new scrolling list.
2	**List(int rows)** Creates a new scrolling list initialized with the specified number of visible lines.
3	**List(int rows, boolean multipleMode)** Creates a new scrolling list initialized to display the specified number of rows.

Class Methods

<T extends EventListener> T[] getListeners(Class<T> listenerType)

Returns an array of all the objects currently registered as FooListeners upon this List.

S.N.	Method & Description
1	**void add(String item)** Adds the specified item to the end of scrolling list.
2	**void add(String item, int index)** Adds the specified item to the the scrolling list at the position indicated by the index.
3	**void addActionListener(ActionListener l)** Adds the specified action listener to receive action events from this list.
4	**void addItem(String item)** Deprecated. replaced by add(String).
5	**void addItem(String item, int index)** Deprecated. replaced by add(String, int).
6	**void addItemListener(ItemListener l)** Adds the specified item listener to receive item events from this list.
7	**void addNotify()** Creates the peer for the list.
8	**boolean allowsMultipleSelections()** Deprecated. As of JDK version 1.1, replaced by isMultipleMode().
9	**void clear()** Deprecated. As of JDK version 1.1, replaced by removeAll().
10	**int countItems()** Deprecated. As of JDK version 1.1, replaced by getItemCount().
11	**void delItem(int position)** Deprecated. replaced by remove(String) and remove(int).
12	**void delItems(int start, int end)** Deprecated. As of JDK version 1.1, Not for public use in the future. This method is expected to be retained only as a package private method.
13	**void deselect(int index)** Deselects the item at the specified index.
14	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this List.

15	**ActionListener[] getActionListeners()** Returns an array of all the action listeners registered on this list.
16	**String getItem(int index)** Gets the item associated with the specified index.
17	**int getItemCount()** Gets the number of items in the list.
18	**ItemListener[] getItemListeners()** Returns an array of all the item listeners registered on this list.
19	**String[] getItems()** Gets the items in the list.
20	**Dimension getMinimumSize()** Determines the minimum size of this scrolling list.
21	**Dimension getMinimumSize(int rows)** Gets the minumum dimensions for a list with the specified number of rows.
22	**Dimension getPreferredSize()** Gets the preferred size of this scrolling list.
23	**Dimension getPreferredSize(int rows)** Gets the preferred dimensions for a list with the specified number of rows.
24	**int getRows()** Gets the number of visible lines in this list.
25	**int getSelectedIndex()** Gets the index of the selected item on the list,
26	**int[] getSelectedIndexes()** Gets the selected indexes on the list.
27	**String getSelectedItem()** Gets the selected item on this scrolling list.
28	**String[] getSelectedItems()** Gets the selected items on this scrolling list.
29	**Object[] getSelectedObjects()** Gets the selected items on this scrolling list in an array of Objects.
30	**int getVisibleIndex()** Gets the index of the item that was last made visible by the method makeVisible.
31	**boolean isIndexSelected(int index)** Determines if the specified item in this scrolling list is selected.
32	**boolean isMultipleMode()** Determines whether this list allows multiple selections.
33	**boolean isSelected(int index)** Deprecated. As of JDK version 1.1, replaced by isIndexSelected(int).
34	**void makeVisible(int index)** Makes the item at the specified index visible.
35	**Dimension minimumSize()** Deprecated. As of JDK version 1.1, replaced by getMinimumSize().
36	**Dimension minimumSize(int rows)** Deprecated. As of JDK version 1.1, replaced by getMinimumSize(int).
37	**protected String paramString()** Returns the parameter string representing the state of this scrolling list.

38	**Dimension preferredSize()** Deprecated. As of JDK version 1.1, replaced by getPreferredSize().
39	**Dimension preferredSize(int rows)** Deprecated. As of JDK version 1.1, replaced by getPreferredSize(int).
40	**protected void processActionEvent(ActionEvent e)** Processes action events occurring on this component by dispatching them to any registered ActionListener objects.
41	**protected void processEvent(AWTEvent e)** Processes events on this scrolling list.
42	**protected void processItemEvent(ItemEvent e)** Processes item events occurring on this list by dispatching them to any registered ItemListener objects.
43	**void remove(int position)** Removes the item at the specified position from this scrolling list.
44	**void remove(String item)** Removes the first occurrence of an item from the list.
45	**void removeActionListener(ActionListener l)** Removes the specified action listener so that it no longer receives action events from this list.
46	**void removeAll()** Removes all items from this list.
47	**void removeItemListener(ItemListener l)** Removes the specified item listener so that it no longer receives item events from this list.
48	**void removeNotify()** Removes the peer for this list.
49	**void replaceItem(String newValue, int index)** Replaces the item at the specified index in the scrolling list with the new string.
50	**void select(int index)** Selects the item at the specified index in the scrolling list.
51	**void setMultipleMode(boolean b)** Sets the flag that determines whether this list allows multiple selections.
52	**void setMultipleSelections(boolean b)** Deprecated. As of JDK version 1.1, replaced by setMultipleMode(boolean).

Methods Inherited

This class inherits methods from the following classes:

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

List Example

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

AwtControlDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtControlDemo(){
      prepareGUI();
   }

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

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
```

```
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

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

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

   private void showListDemo(){

      headerLabel.setText("Control in action: List");
      final List fruitList = new List(4,false);

      fruitList.add("Apple");
      fruitList.add("Grapes");
      fruitList.add("Mango");
      fruitList.add("Peer");

      final List vegetableList = new List(4,true);

      vegetableList.add("Lady Finger");
      vegetableList.add("Onion");
      vegetableList.add("Potato");
      vegetableList.add("Tomato");

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

```
      showButton.addActionListener(new ActionListener() {

         public void actionPerformed(ActionEvent e) {
            String data = "Fruits Selected: "
               + fruitList.getItem(fruitList.getSelectedIndex());
            data += ", Vegetables selected: ";
            for(String vegetable:vegetableList.getSelectedItems()){
               data += vegetable + " ";
            }
            statusLabel.setText(data);
         }
      });

      controlPanel.add(fruitList);
      controlPanel.add(vegetableList);
      controlPanel.add(showButton);

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

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

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

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

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

Output

AWT TextField Class

The textField component allows the user to edit single line of text. When the user types a key in the text field, the event is sent to the TextField. The key event may be key pressed, Key released or key typed. The key event is passed to the registered KeyListener. It is also possible for an ActionEvent; if the ActionEvent is enabled on the textfield, then ActionEvent may be fired by pressing the return key.

Class Declaration

Following is the declaration for java.awt.TextField class:

```
public class TextField
   extends TextComponent
```

Class Constructors

S.N.	Constructor & Description
1	**TextField()** Constructs a new text field.
2	**TextField(int columns)** Constructs a new empty text field with the specified number of columns.
3	**TextField(String text)** Constructs a new text field initialized with the specified text.
4	**TextField(String text, int columns)** Constructs a new text field initialized with the specified text to be displayed, and wide enough to hold the specified number of columns.

Class Methods

S.N.	Method & Description
1	**void addActionListener(ActionListener l)** Adds the specified action listener to receive action events from this text field.
2	**void addNotify()** Creates the TextField's peer.
3	**boolean echoCharIsSet()** Indicates whether or not this text field has a character set for echoing.
4	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this TextField.
5	**ActionListener[] getActionListeners()** Returns an array of all the action listeners registered on this textfield.
6	**int getColumns()** Gets the number of columns in this text field.
7	**char getEchoChar()** Gets the character that is to be used for echoing.
8	**<T extends EventListener> T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this TextField.
9	**Dimension getMinimumSize()** Gets the minumum dimensions for this text field.
10	**Dimension getMinimumSize(int columns) Gets the minumum dimensions for a text field with the specified number of columns.**
11	**Dimension getPreferredSize()** Gets the preferred size of this text field.
12	**Dimension getPreferredSize(int columns)** Gets the preferred size of this text field with the specified number of columns.
13	**Dimension minimumSize()** Deprecated. As of JDK version 1.1, replaced by getMinimumSize().
14	**Dimension minimumSize(int columns)** Deprecated. As of JDK version 1.1, replaced by getMinimumSize(int).
15	**protected String paramString()** Returns a string representing the state of this TextField.
16	**Dimension preferredSize()** Deprecated. As of JDK version 1.1, replaced by getPreferredSize().
17	**Dimension preferredSize(int columns)** Deprecated. As of JDK version 1.1, replaced by getPreferredSize(int).
18	**protected void processActionEvent(ActionEvent e)** Processes action events occurring on this text field by dispatching them to any registered ActionListener objects.
19	**protected void processEvent(AWTEvent e)** Processes events on this text field.
20	**void removeActionListener(ActionListener l)** Removes the specified action listener so that it no longer receives action events from this text field.

21	**void setColumns(int columns)** Sets the number of columns in this text field.
22	**void setEchoChar(char c)** Sets the echo character for this text field.
23	**void setEchoCharacter(char c)** Deprecated. As of JDK version 1.1, replaced by setEchoChar(char).
24	**void setText(String t)** Sets the text that is presented by this text component to be the specified text.

Methods Inherited

This class inherits methods from the following classes:

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

TextField Example

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

AwtControlDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

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

```
public static void main(String[] args){
   AwtControlDemo  awtControlDemo = new AwtControlDemo();
   awtControlDemo.showTextFieldDemo();
}

private void prepareGUI(){
   mainFrame = new Frame("Java AWT 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 Label();
   headerLabel.setAlignment(Label.CENTER);
   statusLabel = new Label();
   statusLabel.setAlignment(Label.CENTER);
   statusLabel.setSize(350,100);

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

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

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

```
      Label  namelabel= new Label("User ID: ", Label.RIGHT);
      Label  passwordLabel = new Label("Password: ", Label.CENTER);
      final TextField userText = new TextField(6);
      final TextField passwordText = new TextField(6);
      passwordText.setEchoChar('*');

      Button loginButton = new Button("Login");

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

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

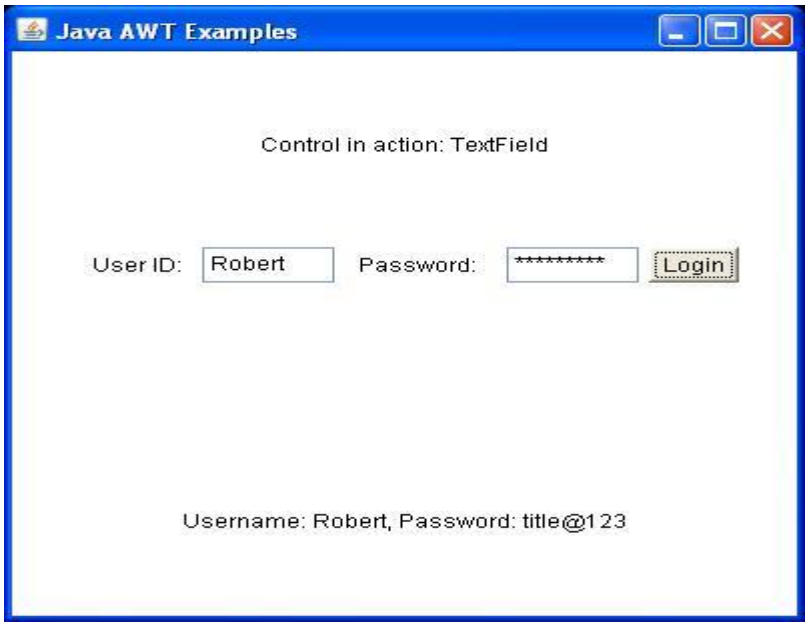
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output:



AWT TextArea Class

The TextArea control in AWT provides us multiline editor area. The user can type here as much as he wants. When the text in the text area becomes larger than the viewable area, the scroll bar appears automatically, which help us to scroll the text up & down and right & left.

Class Declaration

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

```
public class TextArea
   extends TextComponent
```

Field

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

- **static int SCROLLBARS_BOTH** -- Create and display both vertical and horizontal scrollbars.
- **static int SCROLLBARS_HORIZONTAL_ONLY** -- Create and display horizontal scrollbar only.
- **static int SCROLLBARS_NONE** -- Do not create or display any scrollbars for the text area.

- **static int SCROLLBARS_VERTICAL_ONLY** -- Create and display vertical scrollbar only.

Class Constructors

S.N.	Constructor & Description
1	**TextArea()** Constructs a new text area with the empty string as text.
2	**TextArea(int rows, int columns)** Constructs a new text area with the specified number of rows and columns and the empty string as text.
3	**TextArea(String text)** Constructs a new text area with the specified text.
4	**TextArea(String text, int rows, int columns)** Constructs a new text area with the specified text, and with the specified number of rows and columns.
5	**TextArea(String text, int rows, int columns, int scrollbars)** Constructs a new text area with the specified text, and with the rows, columns, and scroll bar visibility as specified.

Class Methods

S.N.	Method & Description
1	**void addNotify()** Creates the TextArea's peer.
2	**void append(String str)** Appends the given text to the text area's current text.
3	**void appendText(String str)** Deprecated. As of JDK version 1.1, replaced by append(String).
4	**AccessibleContext getAccessibleContext()** Returns the AccessibleContext associated with this TextArea.
5	**int getColumns()** Returns the number of columns in this text area.
6	**Dimension getMinimumSize()** Determines the minimum size of this text area.
7	**Dimension getMinimumSize(int rows, int columns)** Determines the minimum size of a text area with the specified number of rows and columns.
8	**Dimension getPreferredSize()** Determines the preferred size of this text area.
9	**Dimension getPreferredSize(int rows, int columns)** Determines the preferred size of a text area with the specified number of rows and columns.
10	**int getRows()** Returns the number of rows in the text area.
11	**int getScrollbarVisibility()** Returns an enumerated value that indicates which scroll bars the text area uses.

12	**void insert(String str, int pos)** Inserts the specified text at the specified position in this text area.
13	**void insertText(String str, int pos)** Deprecated. As of JDK version 1.1, replaced by insert(String, int).
14	**Dimension minimumSize()** Deprecated. As of JDK version 1.1, replaced by getMinimumSize().
15	**Dimension minimumSize(int rows, int columns)** Deprecated. As of JDK version 1.1, replaced by getMinimumSize(int, int).
16	**protected String paramString()** Returns a string representing the state of this TextArea.
17	**Dimension preferredSize()** Deprecated. As of JDK version 1.1, replaced by getPreferredSize().
18	**Dimension preferredSize(int rows, int columns)** Deprecated. As of JDK version 1.1, replaced by getPreferredSize(int, int).
19	**void replaceRange(String str, int start, int end)** Replaces text between the indicated start and end positions with the specified replacement text.
20	**void replaceText(String str, int start, int end)** Deprecated. As of JDK version 1.1, replaced by replaceRange(String, int, int).
21	**void setColumns(int columns)** Sets the number of columns for this text area.
22	**void setRows(int rows)** Sets the number of rows for this text area.

Methods Inherited

This class inherits methods from the following classes:

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

TextArea Example

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

AwtControlDemo.java

```
package com.tutorialspoint.gui;


import java.awt.*;

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

```
public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtControlDemo(){
      prepareGUI();
   }

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

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
```

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

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

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

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

      final TextArea commentTextArea = new TextArea("This is a AWT
tutorial "
      +"to make GUI application in Java.",5,30);

      Button showButton = new Button("Show");

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

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

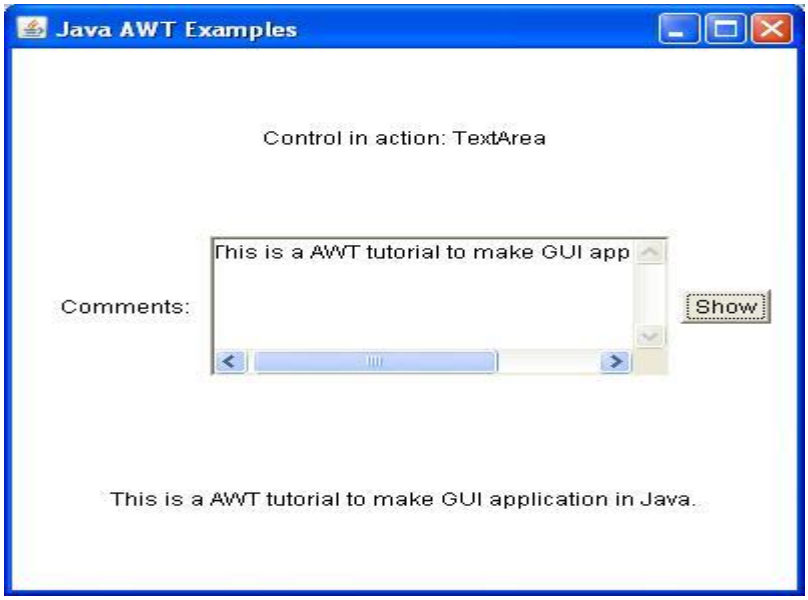
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output:



AWT Choice Class

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

Class Declaration

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

```
public class Choice
   extends Component
      implements ItemSelectable, Accessible
```

Class Constructors

S.N.	Constructor & Description
1	**Choice() ()** Creates a new choice menu.

Class Methods

S.N.	Method & Description
1	**void add(String item)** Adds an item to this Choice menu.
2	**void addItem(String item)** Obsolete as of Java 2 platform v1.1.
3	**void addItemListener(ItemListener l)** Adds the specified item listener to receive item events from this Choice menu.
4	**void addNotify()** Creates the Choice's peer.
5	**int countItems()** Deprecated. As of JDK version 1.1, replaced by getItemCount().
6	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Choice.
7	**String getItem(int index)** Gets the string at the specified index in this Choice menu.
8	**int getItemCount()** Returns the number of items in this Choice menu.
9	**ItemListener[] getItemListeners()** Returns an array of all the item listeners registered on this choice.
10	**<T extends EventListener> T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this Choice.
11	**int getSelectedIndex()** Returns the index of the currently selected item.
12	**String getSelectedItem()** Gets a representation of the current choice as a string.
13	**Object[] getSelectedObjects()** Returns an array (length 1) containing the currently selected item.
14	**void insert(String item, int index)** Inserts the item into this choice at the specified position.
15	**protected String paramString()** Returns a string representing the state of this Choice menu.
16	**protected void processEvent(AWTEvent e)** Processes events on this choice.
17	**protected void processItemEvent(ItemEvent e)** Processes item events occurring on this Choice menu by dispatching them to any registered ItemListener objects.
18	**void remove(int position)** Removes an item from the choice menu at the specified position.
19	**void remove(String item)** Removes the first occurrence of item from the Choice menu.
20	**void removeAll()** Removes all items from the choice menu.
21	**void removeItemListener(ItemListener l)**

	Removes the specified item listener so that it no longer receives item events from this Choice menu.
22	**void select(int pos)** Sets the selected item in this Choice menu to be the item at the specified position.
23	**void select(String str)** Sets the selected item in this Choice menu to be the item whose name is equal to the specified string.

Methods Inherited

This class inherits methods from the following classes:

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

Choice Example

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

AwtControlDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtControlDemo(){
      prepareGUI();
   }

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

```
      AwtControlDemo  awtControlDemo = new AwtControlDemo();
      awtControlDemo.showChoiceDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

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

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

   private void showChoiceDemo(){

      headerLabel.setText("Control in action: Choice");
      final Choice fruitChoice = new Choice();
```

```
      fruitChoice.add("Apple");
      fruitChoice.add("Grapes");
      fruitChoice.add("Mango");
      fruitChoice.add("Peer");

      Button showButton = new Button("Show");

      showButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            String data = "Fruit Selected: "
            + fruitChoice.getItem(fruitChoice.getSelectedIndex());
            statusLabel.setText(data);
         }
      });

      controlPanel.add(fruitChoice);
      controlPanel.add(showButton);

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

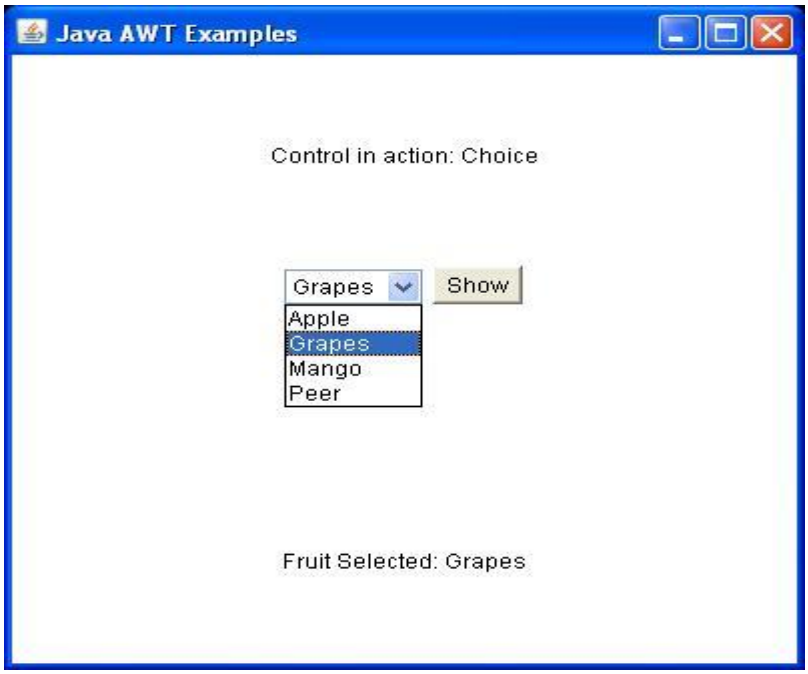
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output:



AWT Canvas Class

Canvas control represents a rectangular area where application can draw something or can receive inputs created by user.

Class Declaration

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

```
public class Canvas
   extends Component
      implements Accessible
```

Class Constructors

S.N.	Constructor & Description
1	**Canvas()** Constructs a new Canvas.
2	**Canvas(GraphicsConfiguration config)** Constructs a new Canvas given a GraphicsConfiguration object.

Class Methods

S.N.	Method & Description
1	**void addNotify()** br>Creates the peer of the canvas.

2	**void createBufferStrategy(int numBuffers)** Creates a new strategy for multi-buffering on this component.
3	**void createBufferStrategy(int numBuffers, BufferCapabilities caps)** Creates a new strategy for multi-buffering on this component with the required buffer capabilities.
4	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Canvas.
5	**BufferStrategy getBufferStrategy()** Returns the BufferStrategy used by this component.
6	**void paint(Graphics g)** Paints this canvas.
7	**void pdate(Graphics g)** Updates this canvas.

Methods Inherited

This class inherits methods from the following classes:

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

Canvas Example

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

AwtControlDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtControlDemo(){
```

```
      prepareGUI();
   }

   public static void main(String[] args){
      AwtControlDemo  awtControlDemo = new AwtControlDemo();
      awtControlDemo.showCanvasDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

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

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

   private void showCanvasDemo(){
```

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

      controlPanel.add(new MyCanvas());
      mainFrame.setVisible(true);
   }

   class MyCanvas extends Canvas {

      public MyCanvas () {
         setBackground (Color.GRAY);
         setSize(300, 300);
      }

      public void paint (Graphics g) {
         Graphics2D g2;
         g2 = (Graphics2D) g;
         g2.drawString ("It is a custom canvas area", 70, 70);
      }
   }
}
```

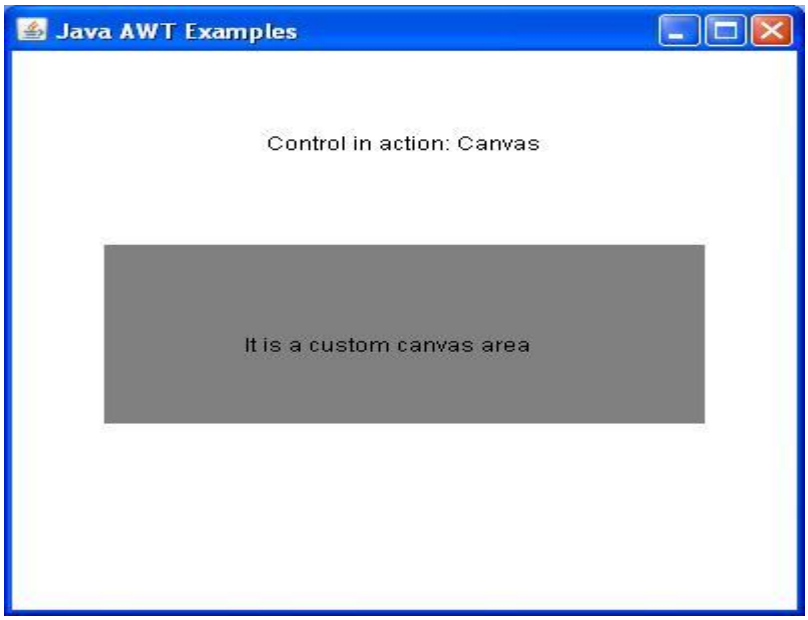
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output:



AWT Image Class

Image control is superclass for all image classes representing graphical images.

Class Declaration

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

```
public abstract class Image
   extends Object
```

Field

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

- **protected float accelerationPriority** -- Priority for accelerating this image.
- **static int SCALE_AREA_AVERAGING** -- Use the Area Averaging image scaling algorithm.
- **static int SCALE_DEFAULT** -- Use the default image-scaling algorithm.
- **static int SCALE_FAST** -- Choose an image-scaling algorithm that gives higher priority to scaling speed than smoothness of the scaled image.
- **static int SCALE_REPLICATE** -- Use the image scaling algorithm embodied in the ReplicateScaleFilter class.

- **static int SCALE_SMOOTH** -- Choose an image-scaling algorithm that gives higher priority to image smoothness than scaling speed.
- **static Object UndefinedProperty** -- The UndefinedProperty object should be returned whenever a property which was not defined for a particular image is fetched.

Class Constructors

S.N.	Constructor & Description
1	**Image()**

Class Methods

S.N.	Method & Description
1	**void flush()** Flushes all reconstructable resources being used by this Image object.
2	**float getAccelerationPriority()** Returns the current value of the acceleration priority hint.
3	**ImageCapabilities getCapabilities(GraphicsConfiguration gc)** Returns an ImageCapabilities object, which can be inquired as to the capabilities of this Image on the specified GraphicsConfiguration.
4	**abstract Graphics getGraphics()** Creates a graphics context for drawing to an off-screen image.
5	**abstract int getHeight(ImageObserver observer)** Determines the height of the image.
6	**abstract Object getProperty(String name, ImageObserver observer)** Gets a property of this image by name.
7	**Image getScaledInstance(int width, int height, int hints)** Creates a scaled version of this image.
8	**abstract ImageProducer getSource()** Gets the object that produces the pixels for the image.
9	**abstract int getWidth(ImageObserver observer)** Determines the width of the image.
10	**void setAccelerationPriority(float priority)** Sets a hint for this image about how important acceleration is.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

Image Example

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

AwtControlDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtControlDemo  awtControlDemo = new AwtControlDemo();
      awtControlDemo.showImageDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
```

```
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

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

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

   private void showImageDemo(){
      headerLabel.setText("Control in action: Image");

      controlPanel.add(new ImageComponent("resources/java.jpg"));
      mainFrame.setVisible(true);
   }

   class ImageComponent extends Component {

      BufferedImage img;

      public void paint(Graphics g) {
         g.drawImage(img, 0, 0, null);
      }

      public ImageComponent(String path) {
         try {
            img = ImageIO.read(new File(path));
         } catch (IOException e) {
```

```
            e.printStackTrace();
         }
      }

      public Dimension getPreferredSize() {
         if (img == null) {
            return new Dimension(100,100);
         } else {
            return new Dimension(img.getWidth(), img.getHeight());
         }
      }
   }
}
```

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

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

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

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

Output

AWT Scrollbar Class

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

Class Declaration

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

```
public class Scrollbar
   extends Component
      implements Adjustable, Accessible
```

Field

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

- **static int HORIZONTAL** --A constant that indicates a horizontal scroll bar.
- **static int VERTICAL** --A constant that indicates a vertical scroll bar.

Class Constructors

S.N.	Constructor & Description
1	**Scrollbar()** Constructs a new vertical scroll bar.
2	**Scrollbar(int orientation)** Constructs a new scroll bar with the specified orientation.
3	**Scrollbar(int orientation, int value, int visible, int minimum, int maximum)** Constructs a new scroll bar with the specified orientation, initial value, visible amount, and minimum and maximum values.

Class Methods

S.N.	Method & Description
1	**void addAdjustmentListener(AdjustmentListener l)** Adds the specified adjustment listener to receive instances of AdjustmentEvent from this scroll bar.
2	**void addNotify()** Creates the Scrollbar's peer.
3	**int getBlockIncrement()** Gets the block increment of this scroll bar.
4	**int getLineIncrement()** Deprecated. As of JDK version 1.1, replaced by getUnitIncrement().
5	**int getMaximum()** Gets the maximum value of this scroll bar.
6	**int getMinimum()**

	Gets the minimum value of this scroll bar.
7	**int getOrientation()** Returns the orientation of this scroll bar.
8	**int getPageIncrement()** Deprecated. As of JDK version 1.1, replaced by getBlockIncrement().
9	**int getUnitIncrement()** Gets the unit increment for this scrollbar.
10	**int getValue()** Gets the current value of this scroll bar.
11	**boolean** getValueIsAdjusting() Returns true if the value is in the process of changing as a result of actions being taken by the user.
12	**int getVisible()** Deprecated. As of JDK version 1.1, replaced by getVisibleAmount().
13	**int getVisibleAmount()** Gets the visible amount of this scroll bar.
14	**protected String paramString()** Returns a string representing the state of this Scrollbar.
15	**protected void processAdjustmentEvent(AdjustmentEvent e)** Processes adjustment events occurring on this scrollbar by dispatching them to any registered AdjustmentListener objects.
16	**protected void processEvent(AWTEvent e)** Processes events on this scroll bar.
17	**void removeAdjustmentListener(AdjustmentListener l)** Removes the specified adjustment listener so that it no longer receives instances of AdjustmentEvent from this scroll bar.
18	**void setBlockIncrement(int v)** Sets the block increment for this scroll bar.
19	**void setLineIncrement(int v)** Deprecated. As of JDK version 1.1, replaced by setUnitIncrement(int).
20	**void setMaximum(int newMaximum)** Sets the maximum value of this scroll bar.
21	**void setMinimum(int newMinimum)** Sets the minimum value of this scroll bar.
22	**void setOrientation(int orientation)** Sets the orientation for this scroll bar.
23	**void setPageIncrement(int v)** Deprecated. As of JDK version 1.1, replaced by setBlockIncrement().
24	**void setUnitIncrement(int v)** Sets the unit increment for this scroll bar.
25	**void setValue(int newValue)** Sets the value of this scroll bar to the specified value.
26	**void setValueIsAdjusting(boolean b)** Sets the valueIsAdjusting property.
27	**void setValues(int value, int visible, int minimum, int maximum)** Sets the values of four properties for this scroll bar: value, visibleAmount, minimum, and maximum.
28	**void setVisibleAmount(int newAmount)**

	Sets the visible amount of this scroll bar.
29	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Scrollbar.
30	**AdjustmentListener[] getAdjustmentListeners()** Returns an array of all the adjustment listeners registered on this scrollbar.
31	**<T extends EventListener>T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this Scrollbar.

Methods Inherited

This class inherits methods from the following classes:

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

Choice Example

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

AwtControlDemo

```
package com.tutorialspoint.gui;

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

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

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

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

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

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

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

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

```
      final Scrollbar horizontalScroller = new
Scrollbar(Scrollbar.HORIZONTAL);
      final Scrollbar verticalScroller = new Scrollbar();
      verticalScroller.setOrientation(Scrollbar.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);
   }
}
```

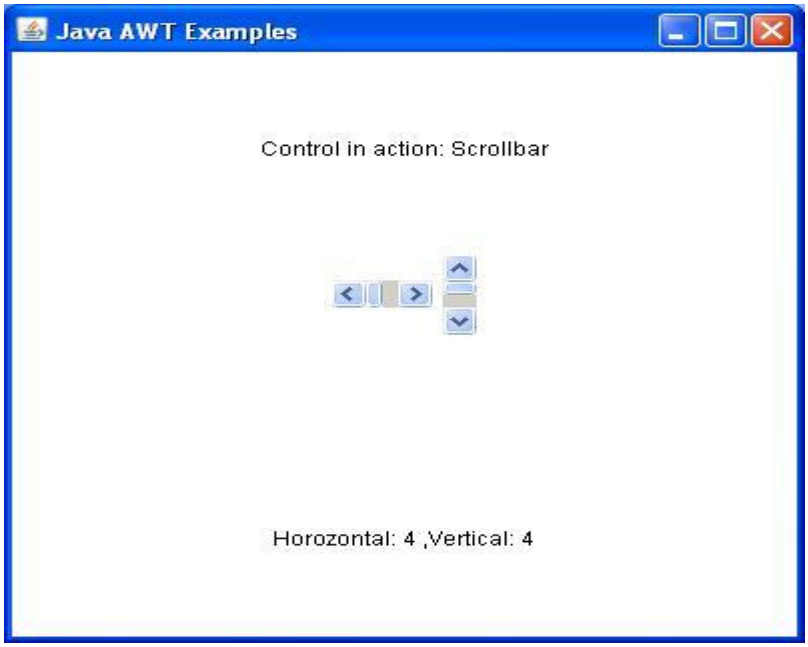
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT Dialog Class

Dialog control represents a top-level window with a title and a border used to take some form of input from the user.

Class Declaration

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

```
public class Dialog
```

```
extends Window
```

Field

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

- static Dialog.ModalityType DEFAULT_MODALITY_TYPE -- Default modality type for modal dialogs.

Class Constructors

S.N.	Constructor & Description
1	**Dialog(Dialog owner)** Constructs an initially invisible, modeless Dialog with the specified owner Dialog and an empty title.
2	**Dialog(Dialog owner, String title)** Constructs an initially invisible, modeless Dialog with the specified owner Dialog and title.
3	**Dialog(Dialog owner, String title, boolean modal)** Constructs an initially invisible Dialog with the specified owner Dialog, title, and modality.
4	**Dialog(Dialog owner, String title, boolean modal, GraphicsConfiguration gc)** Constructs an initially invisible Dialog with the specified owner Dialog, title, modality and GraphicsConfiguration.
5	**Dialog(Frame owner)** Constructs an initially invisible, modeless Dialog with the specified owner Frame and an empty title.
6	**Dialog(Frame owner, boolean modal)** Constructs an initially invisible Dialog with the specified owner Frame and modality and an empty title.
7	**Dialog(Frame owner, String title)** Constructs an initially invisible, modeless Dialog with the specified owner Frame and title.
8	**Dialog(Frame owner, String title, boolean modal)** Constructs an initially invisible Dialog with the specified owner Frame, title and modality.
9	**Dialog(Frame owner, String title, boolean modal, GraphicsConfiguration gc)** Constructs an initially invisible Dialog with the specified owner Frame, title, modality, and GraphicsConfiguration.
10	**Dialog(Window owner)** Constructs an initially invisible, modeless Dialog with the specified owner Window and an empty title.
11	**Dialog(Window owner, Dialog.ModalityType modalityType)** Constructs an initially invisible Dialog with the specified owner Window and modality and an empty title.
12	**Dialog(Window owner, String title)**

	Constructs an initially invisible, modeless Dialog with the specified owner Window and title.
13	**Dialog(Window owner, String title, Dialog.ModalityType modalityType)** Constructs an initially invisible Dialog with the specified owner Window, title and modality.
14	**Dialog(Window owner, String title, Dialog.ModalityType modalityType, GraphicsConfiguration gc)** Constructs an initially invisible Dialog with the specified owner Window, title, modality and GraphicsConfiguration.

Class Methods

S.N.	Method & Description
1	**void addNotify()** Makes this Dialog displayable by connecting it to a native screen resource.
2	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Dialog.
3	**Dialog.ModalityType getModalityType()** Returns the modality type of this dialog.
4	**String getTitle()** Gets the title of the dialog.
5	**void hide()** Deprecated. As of JDK version 1.5, replaced by setVisible(boolean).
6	**boolean isModal()** Indicates whether the dialog is modal.
7	**boolean isResizable()** Indicates whether this dialog is resizable by the user.
8	**boolean isUndecorated()** Indicates whether this dialog is undecorated.
9	**protected String paramString()** Returns a string representing the state of this dialog.
10	**void setModal(boolean modal)** Specifies whether this dialog should be modal.
11	**void setModalityType(Dialog.ModalityType type)** Sets the modality type for this dialog.
12	**void setResizable(boolean resizable)** Sets whether this dialog is resizable by the user.
13	**void setTitle(String title)** Sets the title of the Dialog.
14	**void setUndecorated(boolean undecorated)** Disables or enables decorations for this dialog.
15	**void setVisible(boolean b)** Shows or hides this Dialog depending on the value of parameter b.
16	**void show()** Deprecated. As of JDK version 1.5, replaced by setVisible(boolean).
17	**void toBack()**

	If this Window is visible, sends this Window to the back and may cause it to lose focus or activation if it is the focused or active Window.

Methods Inherited

This class inherits methods from the following classes:

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

Dialog Example

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

AwtControlDemo.java

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

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

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtControlDemo  awtControlDemo = new AwtControlDemo();
      awtControlDemo.showDialogDemo();
   }
```

```
private void prepareGUI(){
   mainFrame = new Frame("Java AWT 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 Label();
   headerLabel.setAlignment(Label.CENTER);
   statusLabel = new Label();
   statusLabel.setAlignment(Label.CENTER);
   statusLabel.setSize(350,100);

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

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

private void showDialogDemo(){
   headerLabel.setText("Control in action: Dialog");
   Button showAboutDialogButton = new Button("Show About Dialog");
   showAboutDialogButton.addActionListener(new ActionListener() {
       @Override
      public void actionPerformed(ActionEvent e) {
         AboutDialog aboutDialog = new AboutDialog(mainFrame);
         aboutDialog.setVisible(true);
```

```
        }
    });

    controlPanel.add(showAboutDialogButton);
    mainFrame.setVisible(true);
}

class AboutDialog extends Dialog {
    public AboutDialog(Frame parent){
        super(parent, true);
        setBackground(Color.gray);
        setLayout(new BorderLayout());
        Panel panel = new Panel();
        panel.add(new Button("Close"));
        add("South", panel);
        setSize(200,200);

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent windowEvent){
                dispose();
            }
        });
    }

    public boolean action(Event evt, Object arg){
        if(arg.equals("Close")){
            dispose();
            return true;
        }
        return false;
    }

    public void paint(Graphics g){
```

```
         g.setColor(Color.white);
         g.drawString("TutorialsPoint.Com", 25,70 );
         g.drawString("Version 1.0", 60, 90);
      }
   }
}
```

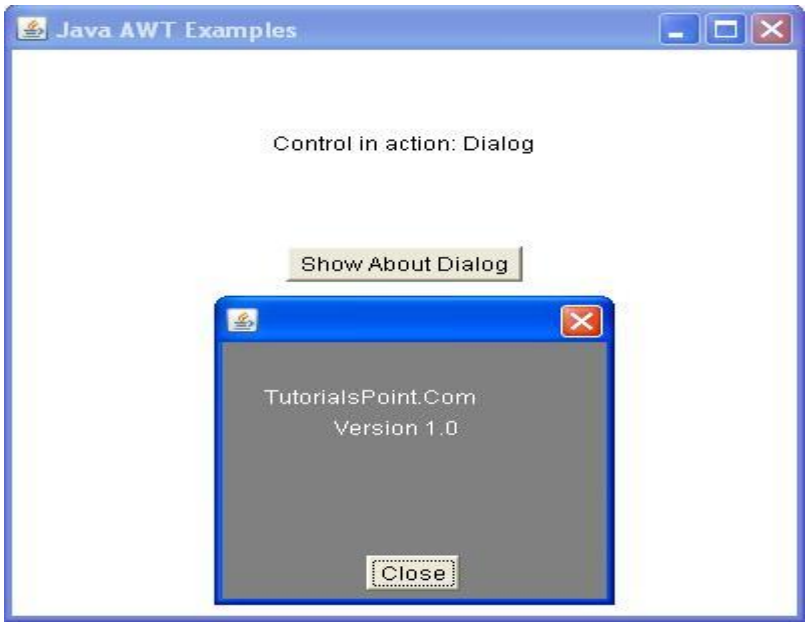
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT FileDialog Class

FileDialog control represents a dialog window from which the user can select a file.

Class Declaration

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

```
public class FileDialog
   extends Dialog
```

Field

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

- **static int LOAD** -- This constant value indicates that the purpose of the file dialog window is to locate a file from which to read.
- **static int SAVE** -- This constant value indicates that the purpose of the file dialog window is to locate a file to which to write.

Class Constructors

S.N.	Constructor & Description
1	**FileDialog(Dialog parent)** Creates a file dialog for loading a file.
2	**FileDialog(Dialog parent, String title)** Creates a file dialog window with the specified title for loading a file.
3	**FileDialog(Dialog parent, String title, int mode)** Creates a file dialog window with the specified title for loading or saving a file.
4	**FileDialog(Frame parent)** Creates a file dialog for loading a file.
5	**FileDialog(Frame parent, String title)** Creates a file dialog window with the specified title for loading a file.
6	**FileDialog(Frame parent, String title, int mode)** Creates a file dialog window with the specified title for loading or saving a file.

Class Methods

S.N.	Method & Description
1	**void addNotify()** Creates the file dialog's peer.
2	**String getDirectory()** Gets the directory of this file dialog.
3	**String getFile()** Gets the selected file of this file dialog.
4	**FilenameFilter getFilenameFilter()** Determines this file dialog's filename filter.
5	**int getMode()** Indicates whether this file dialog box is for loading from a file or for saving to a file.
6	**protected String paramString()** Returns a string representing the state of this FileDialog window.
7	**void setDirectory(String dir)**

	Sets the directory of this file dialog window to be the specified directory.
8	**void setFile(String file)** Sets the selected file for this file dialog window to be the specified file.
9	**void setFilenameFilter(FilenameFilter filter)** Sets the filename filter for this file dialog window to the specified filter.
10	**void setMode(int mode)** Sets the mode of the file dialog.

Methods Inherited

This class inherits methods from the following classes:

- java.awt.Dialog
- java.awt.Window
- java.awt.Component
- java.lang.Object

FileDialog Example

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

AwtControlDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

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

```
   public static void main(String[] args){
      AwtControlDemo  awtControlDemo = new AwtControlDemo();
      awtControlDemo.showFileDialogDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

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

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

   private void showFileDialogDemo(){
      headerLabel.setText("Control in action: FileDialog");
```

```
      final FileDialog fileDialog = new FileDialog(mainFrame,"Select
file");
      Button showFileDialogButton = new Button("Open File");
      showFileDialogButton.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            fileDialog.setVisible(true);
            statusLabel.setText("File Selected :"
            + fileDialog.getDirectory() + fileDialog.getFile());
         }
      });

      controlPanel.add(showFileDialogButton);

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

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

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

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

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

Output

4. AWT – EVENT HANDLING

What is an Event?

Change in the state of an object is known as an **event**. An event describes the change in the state of a 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, scrolling the page are the activities that causes an event to happen.

Types of Event

Events can be broadly classified into two categories:

- **Foreground Events** – Foreground Events are those events, which require the direct interaction of a user. They are generated as consequences of a person interacting with the graphical components in 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** – Background Events are those events, which require the interaction of an end user. Operating system interrupts the hardware or software failure, timer expires, an operation completion are the examples of background events.

What is Event Handling?

Event Handling is a mechanism that controls the event and decides what should happen if an event occurs. This mechanism has the code, which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events. Let's have a brief introduction to this model.

The Delegation Event Model has the following key participants namely:

- **Source** - The source is an object on which event occurs. It is responsible for providing information of the occurred event to its handler. Java provides as with classes for the source object.
- **Listener** - It is also known as event handler. Listener is responsible for generating a response to the event. From java implementation point of view, the listener is also an object. Listener waits until it receives an event. Once the event is received, the listener processes the event and then returns.

The benefit of this approach is that the user interface logic is completely separated from the logic that generates the event. The user interface element is able to delegate the processing of an event to the separate piece of code. In this model, 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 listener that want to receive them.

Steps Involved in Event Handling

- The User clicks the button and the event is generated.
- Now the object of concerned event class is generated and information about the source and the event get populated with the same object.
- Event object is forwarded to the method of registered listener class.
- The method is then executed and returns.

Points to Remember about Listener

- In order to design a listener class, we need 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 if the predefined interfaces, then your class cannot act as a listener class for the source object.

Callback Methods

These are the methods, which are provided by API provider and defined by the application programmer and invoked by the application developer. Here, the callback methods represents 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 that some listener should listen to its events, then 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:/ > AWT > com > tutorialspoint > gui >**

AwtControlDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtControlDemo {

   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtControlDemo(){
      prepareGUI();
   }

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

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
```

```
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      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");

      Button okButton = new Button("OK");
      Button submitButton = new Button("Submit");
      Button cancelButton = new Button("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.");
         }
      }
   }
}
```

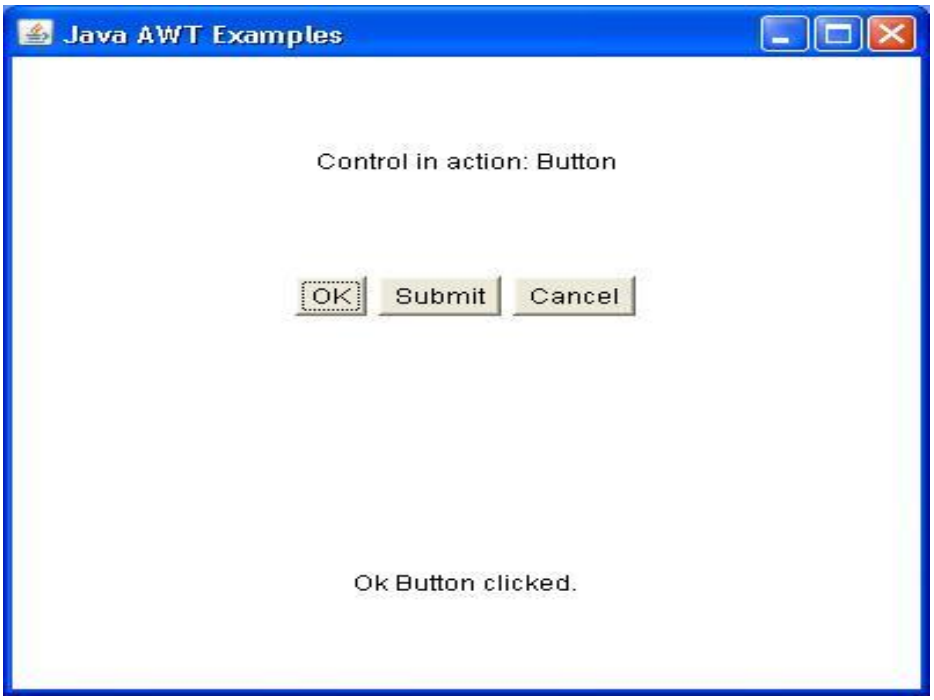
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



5. AWT – EVENT CLASSES

The Event classes represent the event. Java provides us various Event classes, but we will discuss those which are more frequently used.

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**, which is logically deemed to be the object upon which the Event in question initially occurred. 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

S.N.	Constructor & Description
1	**EventObject(Object source)** Constructs a prototypical Event.

Class Methods

S.N.	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 classes:

- java.lang.Object

AWT 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 AWT events. This class and its subclasses supercede the original java.awt.Event class.
2	**ActionEvent** The ActionEvent is generated when button is clicked or the item of a list is double clicked.
3	**InputEvent** The InputEvent class is 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	**TextEvent** The object of this class represents the text events.
7	**WindowEvent** The object of this class represents the change in state of a window.
8	**AdjustmentEvent** The object of this class represents the adjustment event emitted by Adjustable objects.
9	**ComponentEvent** The object of this class represents the change in state of a window.
10	**ContainerEvent** The object of this class represents the change in state of a window.
11	**MouseMotionEvent** The object of this class represents the change in state of a window.
12	**PaintEvent** The object of this class represents the change in state of a window.

AWT AWTEvent Class

It is the root event class for all AWT events. This class and its sub-classes supersede the original java.awt.Event class. This class is defined in java.awt package. This class has a method named getID(), which 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 AWT 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

S.N.	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

S.N.	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 is Consumed()** 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

AWT 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 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

S.N.	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

S.N.	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

AWT InputEvent Class

The InputEvent class is a root event class for all the component-level input events. Input events are delivered to listeners before they are processed normally by the source where they originated. This allows 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

S.N.	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 time stamp of 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

AWT 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. These key events are:

- 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

S.N.	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

S.N.	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

AWT MouseEvent Class

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

- 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
- the 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

S.N.	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

S.N.	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

AWT TextEvent Class

The object of this class represents the text events. The TextEvent is generated when character is entered in the text fields or text area. The TextEvent instance does not include the characters currently in the text component that generated the event rather we are provided with other methods to retrieve that information.

Class Declaration

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

```
public class TextEvent
   extends AWTEvent
```

Field

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

- **static int TEXT_FIRST** --The first number in the range of ids used for text events.
- **static int TEXT_LAST** --The last number in the range of ids used for text events.
- **static int TEXT_VALUE_CHANGED** --This event id indicates that object's text changed.

Class Constructors

S.N.	Constructor & Description
1	**TextEvent(Object source, int id)** Constructs a TextEvent object.

Class Methods

S.N.	Method & Description
1	**String paramString()** Returns a parameter string identifying this text event.

Methods Inherited

This class inherits methods from the following classes:

- java.awt.AWTEvent

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

AWT 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 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

S.N.	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

S.N.	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

AWT AdjustmentEvent Class

The Class **AdjustmentEvent** represents adjustment event emitted by an Adjustable Object.

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

S.N.	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

S.N.	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 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

AWT ComponentEvent Class

The Class **ComponentEvent** represents that a component 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 changed.
- **static int COMPONENT_RESIZED** -- This event indicates that the component's size changed.
- **static int COMPONENT_SHOWN** -- This event indicates that the component was made visible.

Class Constructors

S.N.	Constructor & Description
1	**ComponentEvent(Component source, int id)** Constructs a ComponentEvent object.

Class Methods

S.N.	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

AWT ContainerEvent Class

The Class **ContainerEvent** represents that a container's contents changed because a component was either 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

S.N.	Constructor & Description
1	**ContainerEvent(Component source, int id, Component child)** Constructs a ContainerEvent object.

Class Methods

S.N.	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

AWT MouseMotionEvent Class

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

Class Declaration

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

```
public class MouseMotionEvent
   extends InputEvent
```

Interface Methods

S.N.	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

AWT PaintEvent Class

The Class **PaintEvent** 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

S.N.	Constructor & Description
1	**PaintEvent(Component source, int id, Rectangle updateRect)** Constructs a PaintEvent object with the specified source component and type.

Class Methods

S.N.	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. AWT – EVENT LISTENERS

The Event listener represents the interfaces, responsible to handle events. Java provides us various Event listener classes, but we will discuss those, which are more frequently used. Every method of an event listener has a single argument as an object, which is a subclass of EventObject class. For example, mouse event listener methods will accept instance of MouseEvent, where MouseEvent derives from an EventObject.

EventListner 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
```

AWT 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	**TextListener** This interface is used for receiving the text events.
7	**WindowListener** This interface is used for receiving the window events.

8	**AdjustmentListener** This interface is used for receiving the adjustment events.
9	**ContainerListener** This interface is used for receiving the container events.
10	**MouseMotionListener** This interface is used for receiving the mouse motion events.
11	**FocusListener** This interface is used for receiving the focus events.

AWT 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

S.N.	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:/ > AWT > com > tutorialspoint > gui >**

AwtListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtListenerDemo(){
      prepareGUI();
   }

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

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
```

```
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      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");

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

      Button okButton = new Button("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.");
```

```
      }
   }
}
```

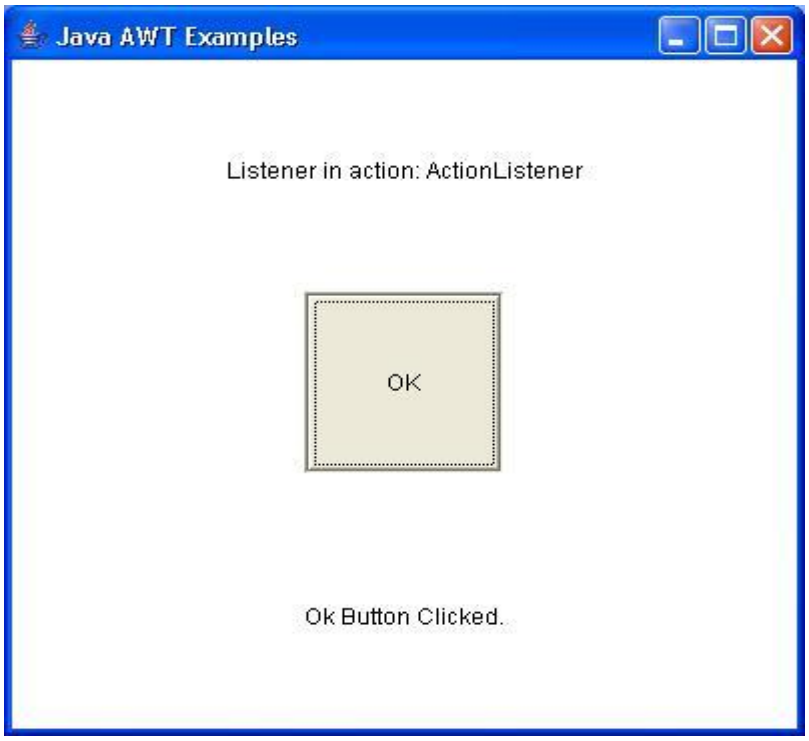
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT 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

S.N.	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 interfaces:

- java.awt.EventListener

ComponentListener Example

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

AwtListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;
```

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

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

private void prepareGUI(){
   mainFrame = new Frame("Java AWT 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 Label();
   headerLabel.setAlignment(Label.CENTER);
   statusLabel = new Label();
   statusLabel.setAlignment(Label.CENTER);
   statusLabel.setSize(350,100);

   controlPanel = new Panel();
   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");

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

   Label msglabel = new Label();
   msglabel.setAlignment(Label.CENTER);
   msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");
   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 command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output

AWT 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

S.N.	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 interfaces:

- java.awt.EventListener

ItemListener Example

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

AwtListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

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

```
public static void main(String[] args){
   AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
   awtListenerDemo.showItemListenerDemo();
}

private void prepareGUI(){
   mainFrame = new Frame("Java AWT 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 Label();
   headerLabel.setAlignment(Label.CENTER);
   statusLabel = new Label();
   statusLabel.setAlignment(Label.CENTER);
   statusLabel.setSize(350,100);

   controlPanel = new Panel();
   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");
   Checkbox chkApple = new Checkbox("Apple");
```

```
      Checkbox chkMango = new Checkbox("Mango");
      Checkbox chkPeer = new Checkbox("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(e.getItem()
         +" Checkbox: "
         + (e.getStateChange()==1?"checked":"unchecked"));
      }
   }
}
```

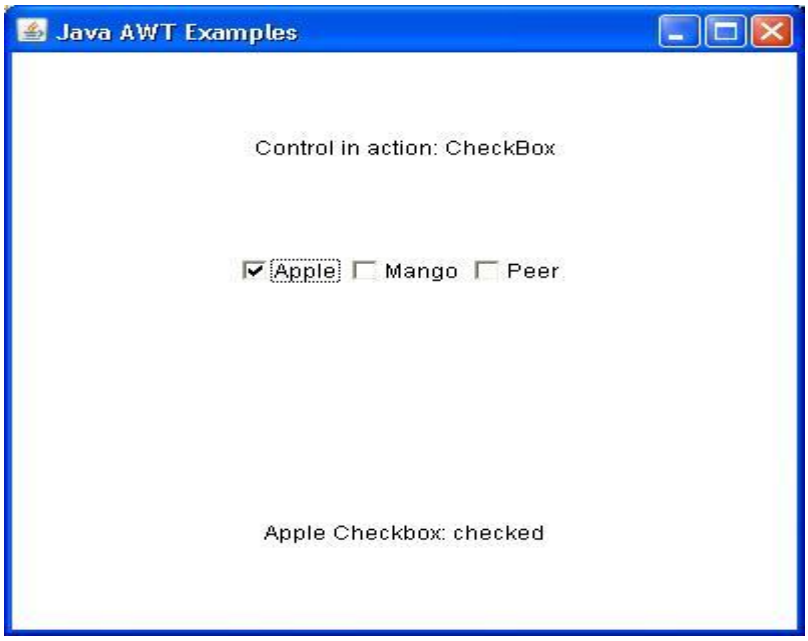
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT 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

S.N.	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 interfaces:

- java.awt.EventListener

KeyListener Example

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

AwtListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;
   private TextField textField;

   public AwtListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
      awtListenerDemo.showKeyListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

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

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

   private void showKeyListenerDemo(){
      headerLabel.setText("Listener in action: KeyListener");

      textField  = new TextField(10);

      textField.addKeyListener(new CustomKeyListener());
      Button okButton = new Button("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) {
      }
   }
}
```

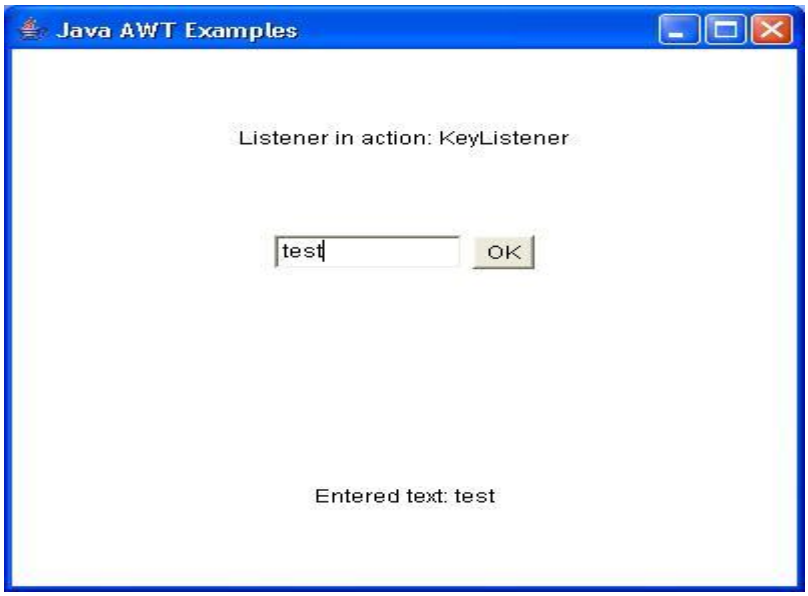
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT 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

S.N.	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 interfaces:

- java.awt.EventListener

MouseListener Example

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

AwtListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
      awtListenerDemo.showMouseListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      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");

      Panel panel = new Panel();
      panel.setBackground(Color.magenta);
      panel.setLayout(new FlowLayout());
      panel.addMouseListener(new CustomMouseListener());

      Label msglabel = new Label();
      msglabel.setAlignment(Label.CENTER);
      msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");
```

```
      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) {
      }
   }
}
```

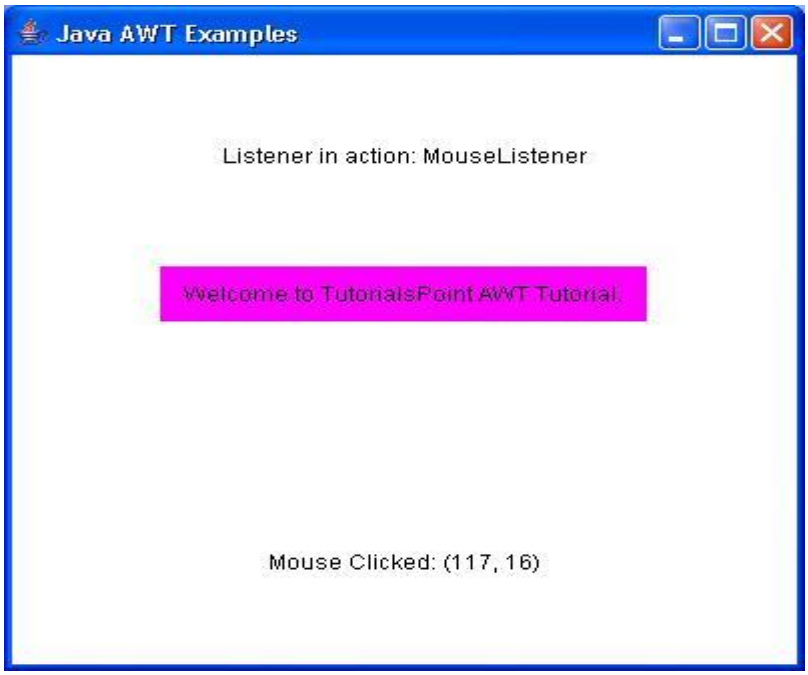
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT TextListener Interface

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

Interface Declaration

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

```
public interface TextListener
   extends EventListener
```

Interface Methods

S.N.	Method & Description
1	**void textValueChanged(TextEvent e)** Invoked when the value of the text has changed.

Methods Inherited

This interface inherits methods from the following interfaces:

- java.awt.EventListener

TextListener Example

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

AwtListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;
   private TextField textField;

   public AwtListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
      awtListenerDemo.showTextListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

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

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

   private void showTextListenerDemo(){
      headerLabel.setText("Listener in action: TextListener");

      textField  = new TextField(10);

      textField.addTextListener(new CustomTextListener());
      Button okButton = new Button("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 CustomTextListener implements TextListener {
      public void textValueChanged(TextEvent e) {
         statusLabel.setText("Entered text: " + textField.getText());
      }
   }
}
```

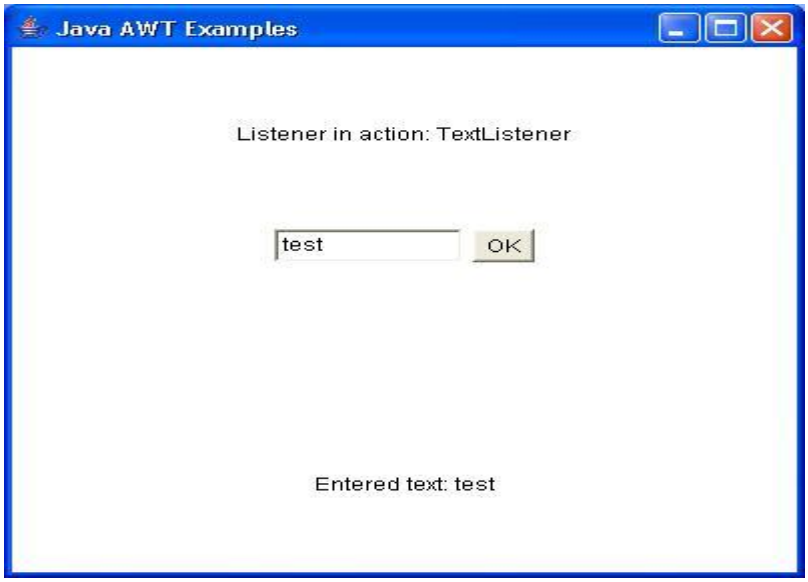
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT 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

S.N.	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 the 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 interfaces:

- java.awt.EventListener

WindowListener Example

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

AwtListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
      awtListenerDemo.showWindowListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
```

```
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      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");

      Button okButton = new Button("OK");

      aboutFrame = new Frame();
      aboutFrame.setSize(300,200);;
      aboutFrame.setTitle("WindowListener Demo");
      aboutFrame.addWindowListener(new CustomWindowListener());

      Label msgLabel = new Label("Welcome to tutorialspoint.");
      msgLabel.setAlignment(Label.CENTER);
      msgLabel.setSize(100,100);
      aboutFrame.add(msgLabel);
      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) {
      }
   }
}
```

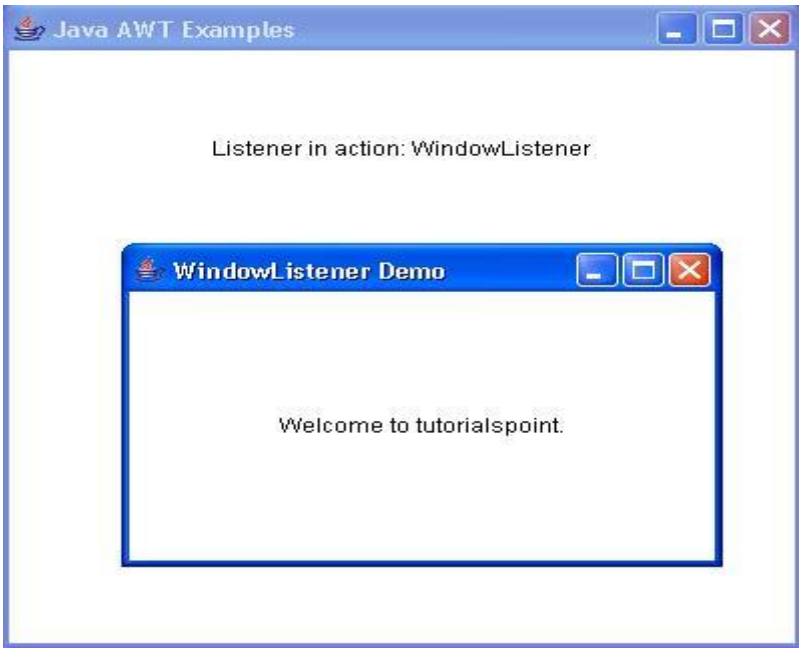
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT AdjustmentListener Interface

The interface**AdjustmentListener** is used for receiving the 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

S.N.	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 interfaces:

- java.awt.event.EventListener

AdjustmentListener Example

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

AwtListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
      awtListenerDemo.showAdjustmentListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      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");

      ScrollPane panel = new ScrollPane();
      panel.setBackground(Color.magenta);
      panel.getHAdjustable().addAdjustmentListener(new
CustomAdjustmentListener());

      Label msglabel = new Label();
      msglabel.setAlignment(Label.CENTER);
      msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");
      panel.add(msglabel);

      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 command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output

AWT ContainerListener Interface

The interface**ContainerListener** is used for receiving the container events. The class that processes container events needs to implement this interface.

Class Declaration

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

```
public interface ContainerListener
extends EventListener
```

Interface Methods

S.N.	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 interfaces:

- java.awt.event.EventListener

ContainerListener Example

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

AwtListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtListenerDemo(){
      prepareGUI();
   }

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

```
      AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
      awtListenerDemo.showContainerListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      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");

      ScrollPane panel = new ScrollPane();
```

```
      panel.setBackground(Color.magenta);
      panel.addContainerListener(new CustomContainerListener());

      Label msglabel = new Label();
      msglabel.setAlignment(Label.CENTER);
      msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");
      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. ");
      }
   }
}
```

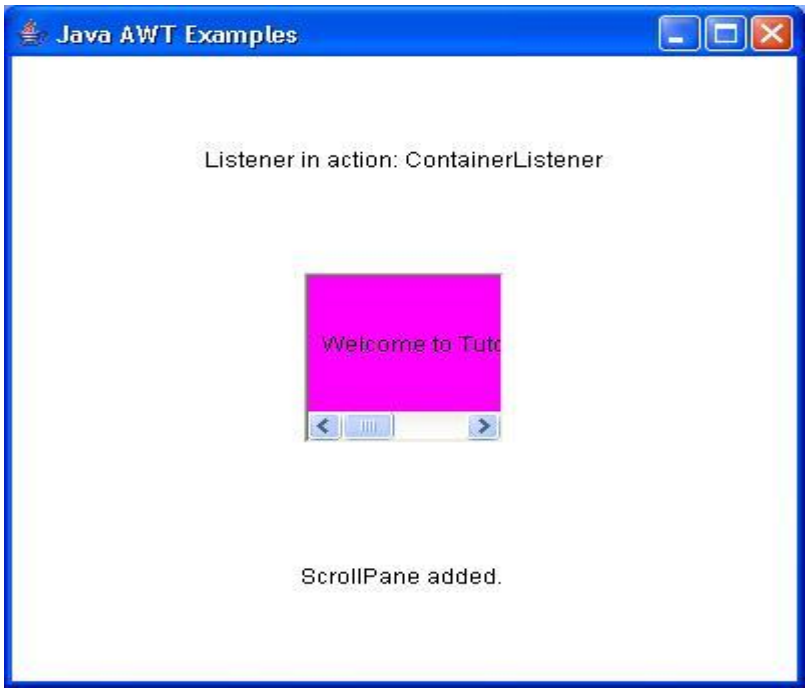
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT MouseMotionListener Interface

The interface**MouseMotionListener** is used for receiving mouse motion events on a component. The class that process mouse motion events needs to implement this interface.

Class Declaration

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

```
public interface MouseMotionListener

extends EventListener
```

Interface Methods

S.N.	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 has been pushed.

Methods Inherited

This class inherits methods from the following interfaces:

- java.awt.event.EventListener

MouseMotionListener Example

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

AwtListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
      awtListenerDemo.showMouseMotionListenerDemo();
   }
   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      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");

      Panel panel = new Panel();
      panel.setBackground(Color.magenta);
      panel.setLayout(new FlowLayout());
      panel.addMouseMotionListener(new CustomMouseMotionListener());

      Label msglabel = new Label();
      msglabel.setAlignment(Label.CENTER);
      msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");
      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() +")");
      }
   }
}
```

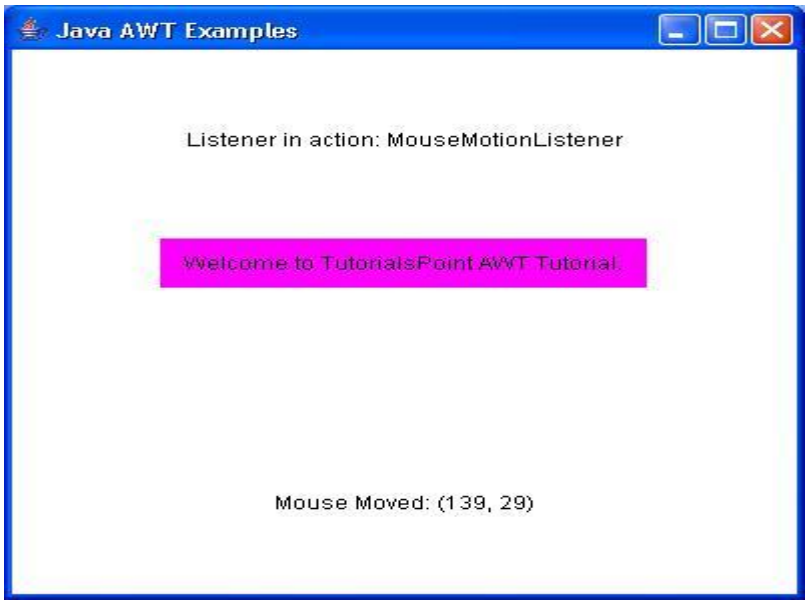
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT FocusListener Interface

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

S.N.	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 interfaces:

- java.awt.event.EventListener

FocusListener Example

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

AwtListenerDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
      awtListenerDemo.showFocusListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      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");

      Button okButton = new Button("OK");
      Button cancelButton = new Button("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. ");
         }
      }
}
```

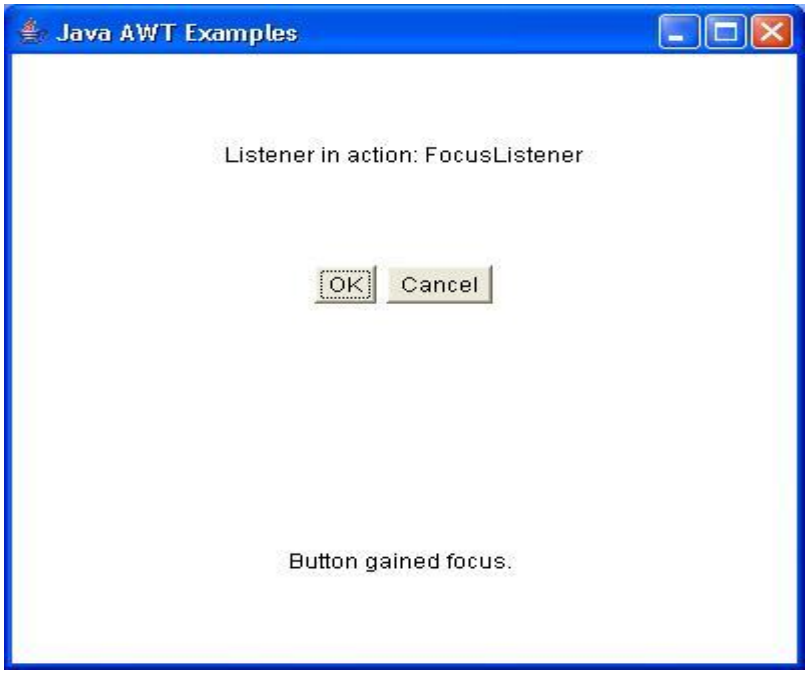
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



7. AWT – 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.

AWT Adapters

Following is the list of commonly used adapters while listening GUI events in AWT.

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.

AWT FocusAdapter Class

The class **FocusAdapter** is an abstract (adapter) class for receiving the keyboard focus events. All methods of this class are empty. This is convenience class for the listener objects.

Class Declaration

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

```
public abstract class FocusAdapter
   extends Object
      implements FocusListener
```

Class Constructors

S.N.	Constructor & Description
1	**FocusAdapter()**

Class Methods

S.N.	Method & Description
1	**void focusGained(FocusEvent e)** Invoked when a component gains the keyboard focus.
2	**focusLost(FocusEvent e)** Invoked when a component loses 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:/ > AWT > com > tutorialspoint > gui >**

AwtAdapterDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtAdapterDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtAdapterDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtAdapterDemo  awtAdapterDemo = new AwtAdapterDemo();
      awtAdapterDemo.showFocusAdapterDemo();
   }
```

```
private void prepareGUI(){
   mainFrame = new Frame("Java AWT 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 Label();
   headerLabel.setAlignment(Label.CENTER);
   statusLabel = new Label();
   statusLabel.setAlignment(Label.CENTER);
   statusLabel.setSize(350,100);

   controlPanel = new Panel();
   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");

   Button okButton = new Button("OK");
   Button cancelButton = new Button("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);
   }
}
```

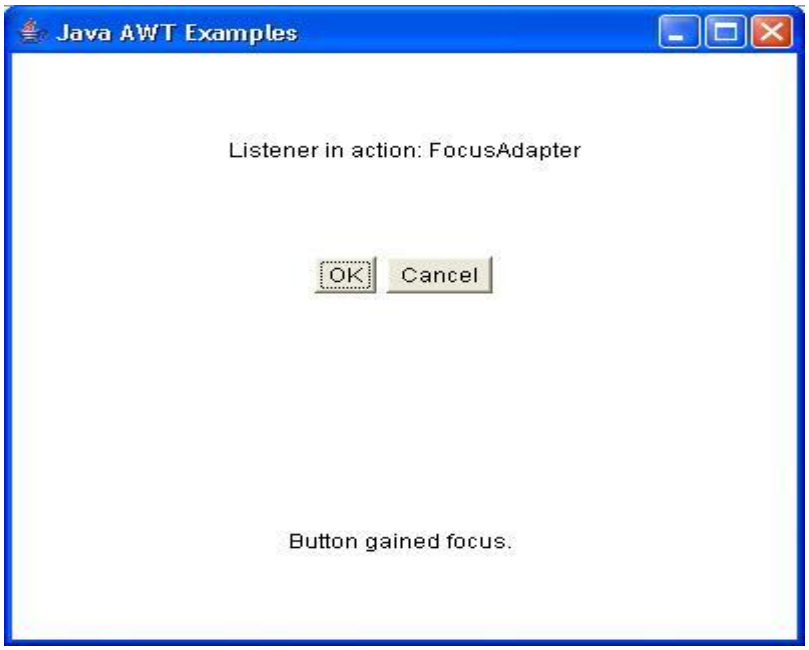
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT KeyAdapter Class

The class **KeyAdapter** is an abstract (adapter) class for receiving the keyboard events. All methods of this class are empty. This is convenience class for creating the listener objects.

Class Declaration

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

```
public abstract class KeyAdapter
   extends Object
      implements KeyListener
```

Class Constructors

S.N.	Constructor & Description
1	**KeyAdapter()**

Class Methods

S.N.	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 classes:

- java.lang.Object

KeyAdapter Example

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

AwtAdapterDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtAdapterDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtAdapterDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtAdapterDemo  awtAdapterDemo = new AwtAdapterDemo();
      awtAdapterDemo.showKeyAdapterDemo();
   }

   private void prepareGUI(){
```

```
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      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 TextField textField = new TextField(10);

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

```
            }
         }
      });
      Button okButton = new Button("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);
   }
}
```

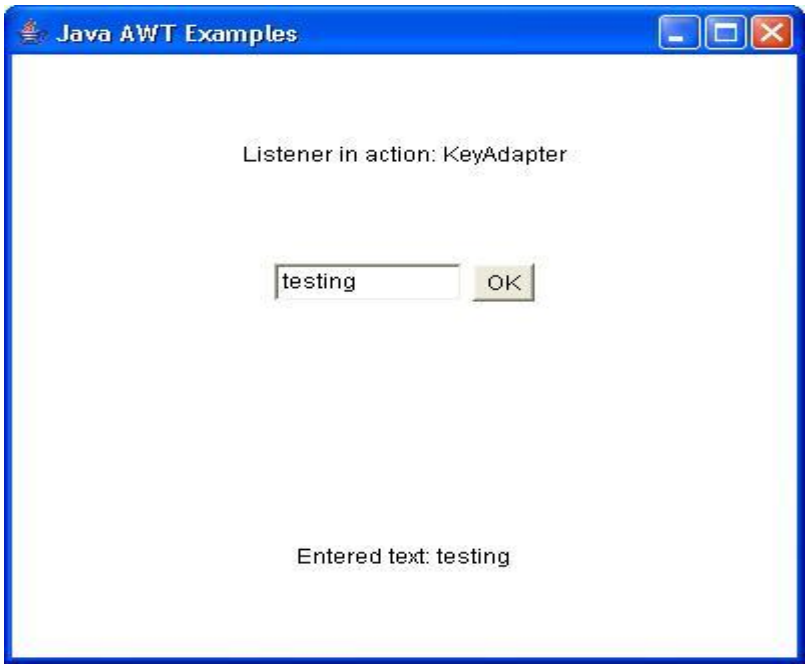
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT MouseAdapter Class

The class **MouseAdapter** is an abstract (adapter) class for receiving the mouse events. All methods of this class are empty. This is convenience class for the 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

S.N.	Constructor & Description
1	**MouseAdapter()**

Class Methods

S.N.	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 classes:

- java.lang.Object

MouseAdapter Example

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

AwtAdapterDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtAdapterDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

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

```
public static void main(String[] args){
   AwtAdapterDemo  awtAdapterDemo = new AwtAdapterDemo();
   awtAdapterDemo.showMouseAdapterDemo();
}

private void prepareGUI(){
   mainFrame = new Frame("Java AWT 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 Label();
   headerLabel.setAlignment(Label.CENTER);
   statusLabel = new Label();
   statusLabel.setAlignment(Label.CENTER);
   statusLabel.setSize(350,100);

   controlPanel = new Panel();
   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");
```

```
      Panel panel = new Panel();
      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() +")");
         }
      });

      Label msglabel = new Label();
      msglabel.setAlignment(Label.CENTER);
      msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");

      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);
   }
}
```

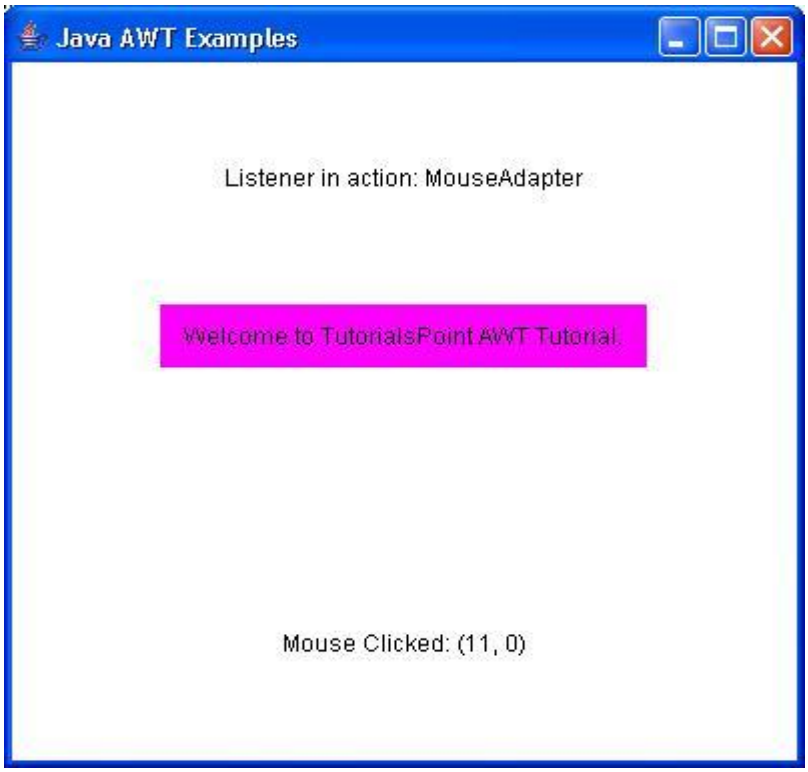
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT MouseMotionAdapter Class

The class **MouseMotionAdapter** is an abstract (adapter) class for receiving the mouse motion events. All methods of this class are empty. This is convenience class for the listener objects.

Class Declaration

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

```
public abstract class MouseMotionAdapter
   extends Object
      implements MouseMotionListener
```

Class Constructors

S.N.	Constructor & Description
1	**MouseMotionAdapter()**

Class Methods

S.N.	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:/ > AWT > com > tutorialspoint > gui >**

AwtAdapterDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtAdapterDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtAdapterDemo(){
      prepareGUI();
   }

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

```
      AwtAdapterDemo  awtAdapterDemo = new AwtAdapterDemo();
      awtAdapterDemo.showMouseMotionAdapterDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      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");

      Panel panel = new Panel();
      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() +")");
         }
      });

      Label msglabel = new Label();
      msglabel.setAlignment(Label.CENTER);
      msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");
      panel.add(msglabel);
      controlPanel.add(panel);

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

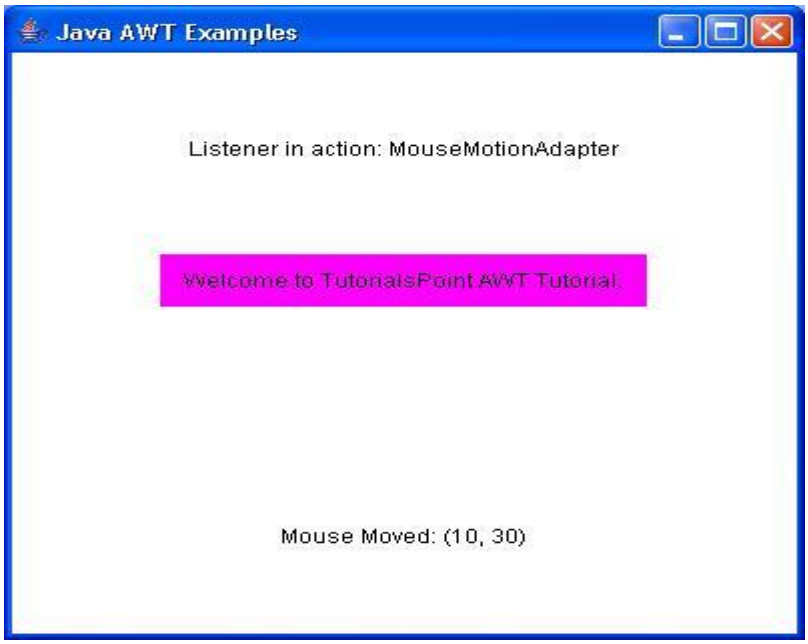
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT WindowAdapter Class

The class **WindowAdapter** is an abstract (adapter) class for receiving the window events. All methods of this class are empty. This is convenience class for the 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

S.N.	Constructor & Description
1	**WindowAdapter()**

Class Methods

S.N.	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 classes:

- java.lang.Object

WindowAdapter Example

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

AwtAdapterDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtAdapterDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
```

```
   private Panel controlPanel;

   public AwtAdapterDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtAdapterDemo  awtAdapterDemo = new AwtAdapterDemo();
      awtAdapterDemo.showWindowAdapterDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      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");

      Button okButton = new Button("OK");

      final Frame aboutFrame = new Frame();
      aboutFrame.setSize(300,200);;
      aboutFrame.setTitle("WindowAdapter Demo");
      aboutFrame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent windowEvent){
               aboutFrame.dispose();
         }
      });
      Label msgLabel = new Label("Welcome to tutorialspoint.");
      msgLabel.setAlignment(Label.CENTER);
      msgLabel.setSize(100,100);
      aboutFrame.add(msgLabel);
      aboutFrame.setVisible(true);
   }
}
```

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

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

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

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

Output

8. AWT – LAYOUTS

Introduction

Layout means the arrangement of components within the container. In other way, we can say that placing the components at a particular position within the container. The task of layouting the controls is done automatically by the Layout Manager.

Layout Manager

The layout manager automatically positions all the components within the container. If we do not use layout manager even then the components are positioned by the default layout manager. It is possible to layout the controls by hand, but it becomes very difficult because of the following two reasons:

- It is very tedious to handle a large number of controls within the container.
- More often, the width and height information of a component is not given when we need to arrange them.

Java provides us various layout manager to position the controls. The properties like size, shape, and arrangement varies from one layout manager to other layout manager. 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 applet viewer 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 Layout Manager interface.

Following are the interfaces defining functionalities of Layout Managers:

Sr. No.	Interface & Description
1	**Layout Manager** The Layout Manager interface declares those methods, which need to be implemented by the class whose object will act as a layout manager.
2	**Layout Manager2** The Layout Manager2 is the sub-interface of the Layout Manager. This interface is for those classes, which know how to layout containers based on layout constraint object.

AWT Layout Manager Interface

Theinterface**LayoutManager** is used to define the interface for classes that know how to lay out the Containers.

Class Declaration

Following is the declaration for **java.awt.LayoutManager** interface:

```
public interface LayoutManager
```

Interface Methods

S.N.	Method & Description
1	**void addLayoutComponent(String name, Component comp)** If the layout manager uses a per-component string, adds the component 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 that it contains.
4	**Dimension preferredLayoutSize(Container parent)** Calculates the preferred size dimensions for the specified container, given the components that it contains.
5	**void removeLayoutComponent(Component comp)** Removes the specified component from the layout.

AWT LayoutManager2 Interface

The **Layout Manger2** Interface is used to define the interface for classes, which 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

S.N.	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 the information that should be discarded.
5	**Dimension maximumLayoutSize(Container target)** Calculates the maximum size dimensions for the specified container, given the components that it contains.

AWT Layout Manager Classes

Following is the list of commonly used controls while designed 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 layouts 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 same size.

AWT BorderLayout Class

The class **BorderLayout** arranges the components to fit in the five regions: east, west, north, south and center. Each region contains 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 container).
- static String EAST -- The east layout constraint (right side of 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 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 container).
- static String WEST -- The west layout constraint (left side of container).

Class Constructors

S.N.	Constructor & Description
1	**BorderLayout()** Constructs a new border layout with no gaps between components.

2	**BorderLayout(int hgap, int vgap)** Constructs a border layout with the specified gaps between components.

Class Methods

S.N.	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 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 components.
7	**void invalidateLayout(Container target)** Invalidates the layout, indicating that if the layout manager has cached the information that 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 components.
14	**void setVgap(int vgap)** Sets the vertical gap between components.
15	**String toString()** Returns a string representation of the state of this border layout.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

BorderLayout Example

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

AwtLayoutDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtLayoutDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;
   private Label msglabel;

   public AwtLayoutDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtLayoutDemo  awtLayoutDemo = new AwtLayoutDemo();
      awtLayoutDemo.showBorderLayoutDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      msglabel = new Label();
      msglabel.setAlignment(Label.CENTER);
      msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");

      controlPanel = new Panel();
      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");

      Panel panel = new Panel();
      panel.setBackground(Color.darkGray);
      panel.setSize(300,300);
      BorderLayout layout = new BorderLayout();
      layout.setHgap(10);
      layout.setVgap(10);
      panel.setLayout(layout);
```

```
      panel.add(new Button("Center"),BorderLayout.CENTER);
      panel.add(new Button("Line Start"),BorderLayout.LINE_START);
      panel.add(new Button("Line End"),BorderLayout.LINE_END);
      panel.add(new Button("East"),BorderLayout.EAST);
      panel.add(new Button("West"),BorderLayout.WEST);
      panel.add(new Button("North"),BorderLayout.NORTH);
      panel.add(new Button("South"),BorderLayout.SOUTH);

      controlPanel.add(panel);

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

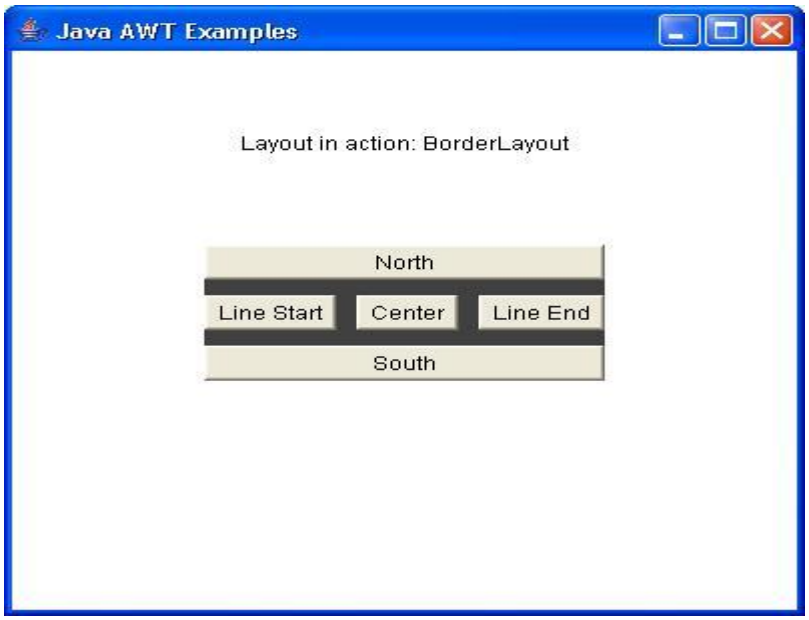
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command.

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

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

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

Output



AWT CardLayout Class

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

S.N.	Constructor & Description
1	**CardLayout()** Creates a new card layout with gaps of size zero.
2	**CardLayout(int hgap, int vgap)** Creates a new card layout with the specified horizontal and vertical gaps.

Class Methods

S.N.	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 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 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 components.
8	**void invalidateLayout(Container target)** Invalidates the layout, indicating that if the layout manager has cached the information that 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 components.
18	**void setVgap(int vgap)** Sets the vertical gap between 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 classes:

- java.lang.Object

CardLayout Example

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

AwtLayoutDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtLayoutDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;
   private Label msglabel;

   public AwtLayoutDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtLayoutDemo  awtLayoutDemo = new AwtLayoutDemo();
      awtLayoutDemo.showCardLayoutDemo();
   }

   private void prepareGUI(){
```

```
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      msglabel = new Label();
      msglabel.setAlignment(Label.CENTER);
      msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");

      controlPanel = new Panel();
      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 Panel panel = new Panel();
      panel.setBackground(Color.CYAN);
      panel.setSize(300,300);
```

```
CardLayout layout = new CardLayout();
layout.setHgap(10);
layout.setVgap(10);
panel.setLayout(layout);

Panel buttonPanel = new Panel(new FlowLayout());

buttonPanel.add(new Button("OK"));
buttonPanel.add(new Button("Cancel"));

Panel textBoxPanel = new Panel(new FlowLayout());

textBoxPanel.add(new Label("Name:"));
textBoxPanel.add(new TextField(20));

panel.add("Button", buttonPanel);
panel.add("Text", textBoxPanel);

Choice choice = new Choice();
choice.add("Button");
choice.add("Text");

choice.addItemListener(new ItemListener() {
   public void itemStateChanged(ItemEvent e) {
      CardLayout cardLayout = (CardLayout)(panel.getLayout());
      cardLayout.show(panel, (String)e.getItem());
   }
});
controlPanel.add(choice);
controlPanel.add(panel);

mainFrame.setVisible(true);
```

```
   }
}
```

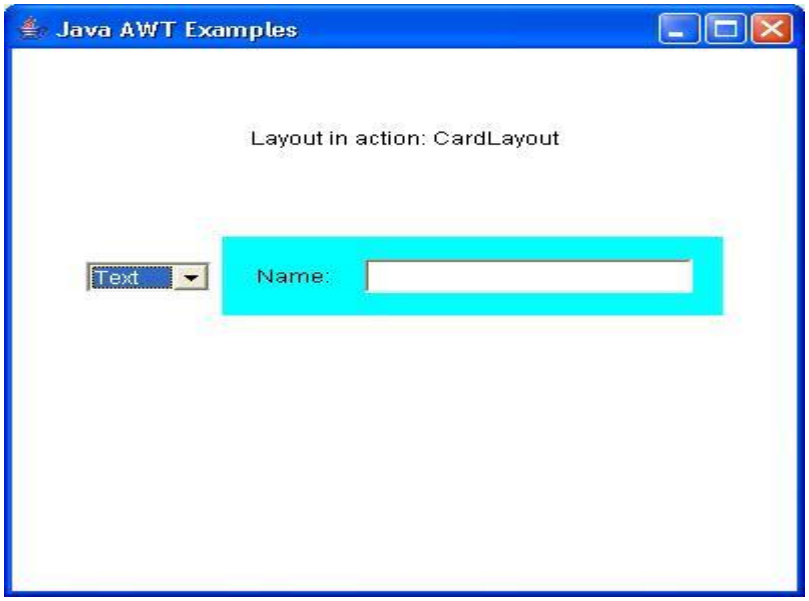
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT FlowLayout Class

The class **FlowLayout** arranges 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

S.N.	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

S.N.	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 components.
4	**int getVgap()** Gets the vertical gap between components.
5	**void layoutContainer(Container target)** Lays out the container.
6	**Dimension minimumLayoutSize(Container target)** Returns the minimum dimensions needed to layout 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 components.
11	**void setVgap(int vgap)** Sets the vertical gap between components.
12	**String toString()** Returns a string representation of this FlowLayout object and its values.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

FlowLayout Example

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

AwtLayoutDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtLayoutDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;
   private Label msglabel;

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

```
public static void main(String[] args){
   AwtLayoutDemo  awtLayoutDemo = new AwtLayoutDemo();
   awtLayoutDemo.showFlowLayoutDemo();
}

private void prepareGUI(){
   mainFrame = new Frame("Java AWT 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 Label();
   headerLabel.setAlignment(Label.CENTER);
   statusLabel = new Label();
   statusLabel.setAlignment(Label.CENTER);
   statusLabel.setSize(350,100);

   msglabel = new Label();
   msglabel.setAlignment(Label.CENTER);
   msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");

   controlPanel = new Panel();
   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");

      Panel panel = new Panel();
      panel.setBackground(Color.darkGray);
      panel.setSize(200,200);
      FlowLayout layout = new FlowLayout();
      layout.setHgap(10);
      layout.setVgap(10);
      panel.setLayout(layout);
      panel.add(new Button("OK"));
      panel.add(new Button("Cancel"));

      controlPanel.add(panel);

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

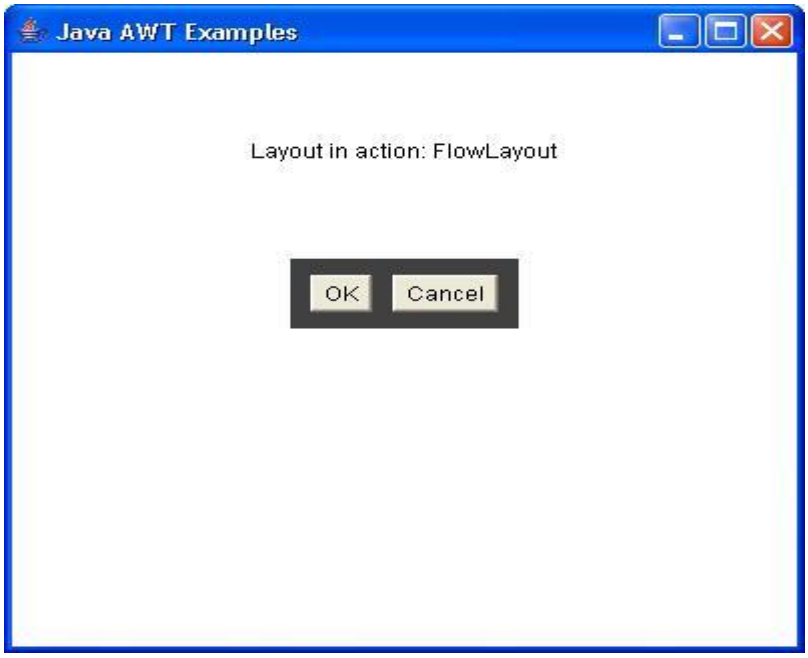
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT GridLayout Class

The class **GridLayout** arranges 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

S.N.	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

S.N.	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 components.
4	**int getRows()** Gets the number of rows in this layout.
5	**int getVgap()** Gets the vertical gap between 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 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 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 classes:

- java.lang.Object

GridLayout Example

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

AwtLayoutDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtLayoutDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;
   private Label msglabel;

   public AwtLayoutDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtLayoutDemo  awtLayoutDemo = new AwtLayoutDemo();
      awtLayoutDemo.showGridLayoutDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
```

```
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      msglabel = new Label();
      msglabel.setAlignment(Label.CENTER);
      msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");

      controlPanel = new Panel();
      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");

      Panel panel = new Panel();
      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 Button("Button 1"));
      panel.add(new Button("Button 2"));
      panel.add(new Button("Button 3"));
      panel.add(new Button("Button 4"));
```

```
      panel.add(new Button("Button 5"));
      controlPanel.add(panel);
      mainFrame.setVisible(true);
   }
}
```

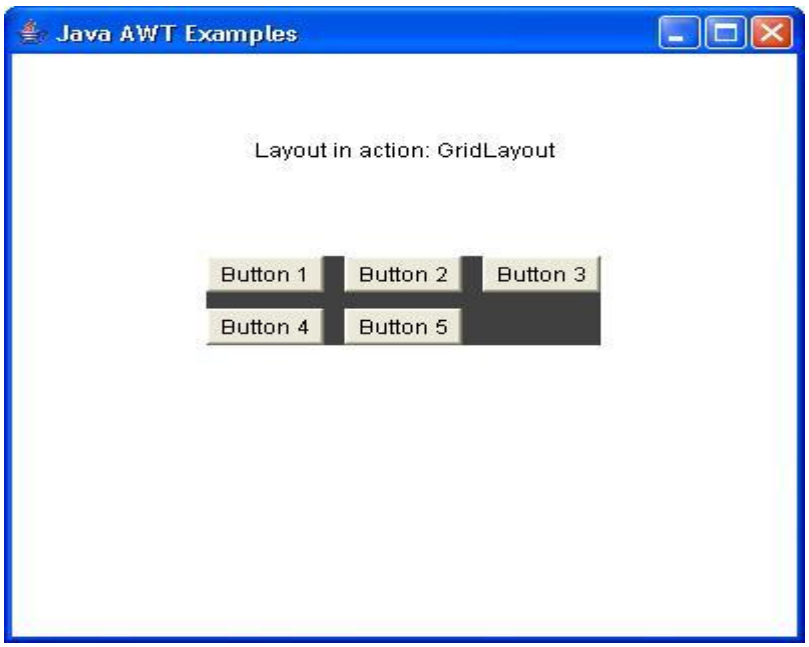
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT GridBagLayout Class

The class **GridBagLayout** arranges 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.BorderLayout** class:

- **double[] columnWeights** -- This field holds the overrides to the column weights.
- **int[] columnWidths** -- This field holds the overrides to the column minimum width.
- **protected Hashtable comptable** -- This hashtable maintains the association between a component and its gridbag constraints.
- **protected GridBagConstraints defaultConstraints** -- This field holds a gridbag constraints instance containing the default values, so if a component does not have gridbag constraints associated with it, then the component will be assigned a copy of the defaultConstraints.
- **protected java.awt.GridBagLayoutInfo layoutInfo** -- This field holds the layout information for the gridbag.
- **protected static int MAXGRIDSIZE** -- The maximum number of grid positions (both horizontally and vertically) that can be laid out by the grid bag layout.
- **protected static int MINSIZE** -- The smallest grid that can be laid out by the grid bag layout.
- **protected static int PREFERREDSIZE** -- The preferred grid size that can be laid out by the grid bag layout.
- **int[] rowHeights** -- This field holds the overrides to the row minimum heights.
- **double[] rowWeights** -- This field holds the overrides to the row weights.Class Constructors

S.N.	Constructor & Description
1	**GridBagLayout()** Creates a grid bag layout manager.

Class Methods

S.N.	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:/ > AWT > com > tutorialspoint > gui >**

AwtLayoutDemo.java

```
package com.tutorialspoint.gui;


import java.awt.*;

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

```
public class AwtLayoutDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;
   private Label msglabel;

   public AwtLayoutDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtLayoutDemo  awtLayoutDemo = new AwtLayoutDemo();
      awtLayoutDemo.showGridBagLayoutDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      msglabel = new Label();
```

```
      msglabel.setAlignment(Label.CENTER);
      msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");

      controlPanel = new Panel();
      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");

      Panel panel = new Panel();
      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 Button("Button 1"),gbc);

      gbc.gridx = 1;
      gbc.gridy = 0;
      panel.add(new Button("Button 2"),gbc);

      gbc.fill = GridBagConstraints.HORIZONTAL;
```

```
      gbc.ipady = 20;
      gbc.gridx = 0;
      gbc.gridy = 1;
      panel.add(new Button("Button 3"),gbc);

      gbc.gridx = 1;
      gbc.gridy = 1;
      panel.add(new Button("Button 4"),gbc);

      gbc.gridx = 0;
      gbc.gridy = 2;
      gbc.fill = GridBagConstraints.HORIZONTAL;
      gbc.gridwidth = 2;
      panel.add(new Button("Button 5"),gbc);

      controlPanel.add(panel);

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

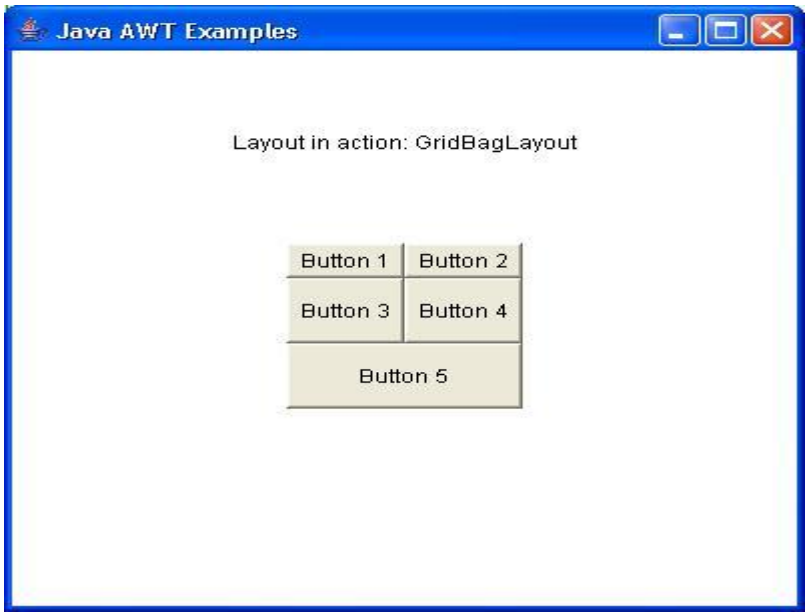
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



9. AWT – CONTAINERS

Containers are integral part of the AWT GUI components. A container provides space where a component can be located. A Container in AWT is a component itself and it adds the capability to add component to itself.

Following are the noticeable points to be considered:

- Sub classes of Container are called as Container. For example Panel, Frame and Window.
- Container can add only Component to itself.
- A default layout is present in each container, which can be overridden using setLayout method.

Sr. No.	Container & Description
1	**Container** It is a generic container object, which can contain other AWT components.

AWT Container Class

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

S.N.	Constructor & Description
1	**Container()** This creates a new Container.

Class Methods

S.N.	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 add Property Change Listener(String proper tyName, Property Change Listener 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 nth 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 indicate 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

AWT UI Elements

Following is the list of commonly used containers while designed GUI using AWT:

Sr. No.	Container & Description
1	**Panel** Panel is the simplest container. It provides space in which any other component can be placed, including other panels.
2	**Frame** A Frame is a top-level window with a title and a border.

3	**Window** A Window object is a top-level window with no border and no menubar.

AWT Panel Class

The class **Panel** is the simplest container class. It provides space in which an application can attach any other component, including other panels. It uses FlowLayout as default layout manager.

Class Declaration

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

```
public class Panel
   extends Container
      implements Accessible
```

Class Constructors

S.N.	Constructor & Description
1	**Panel()** Creates a new panel using the default layout manager.
2	**Panel(LayoutManager layout)** Creates a new panel with the specified layout manager.

Class Methods

S.N.	Method & Description
1	**void addNotify()** Creates the Panel's peer.
2	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Panel.

Methods Inherited

This class inherits methods from the following classes:

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

Panel Example

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

AwtContainerDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtContainerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;
   private Label msglabel;

   public AwtContainerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtContainerDemo  awtContainerDemo = new AwtContainerDemo();
      awtContainerDemo.showPanelDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
```

```
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      msglabel = new Label();
      msglabel.setAlignment(Label.CENTER);
      msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");

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

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

   private void showPanelDemo(){
      headerLabel.setText("Container in action: Panel");

      Panel panel = new Panel();
      panel.setBackground(Color.magenta);
      panel.setLayout(new FlowLayout());
      panel.add(msglabel);

      controlPanel.add(panel);

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

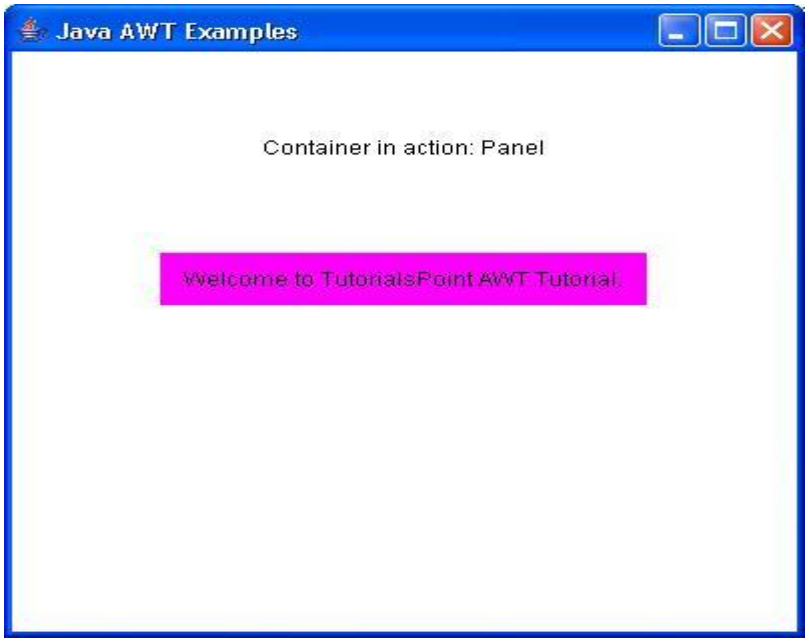
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT Frame Class

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

Class Declaration

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

```
public class Frame
   extends Window
      implements MenuContainer
```

Field

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

- **static float BOTTOM_ALIGNMENT** -- Ease-of-use constant for getAlignmentY.
- **static int CROSSHAIR_CURSOR** -- Deprecated. Replaced by Cursor.CROSSHAIR_CURSOR.
- **static int DEFAULT_CURSOR** -- Deprecated. Replaced by Cursor.DEFAULT_CURSOR.
- **static int E_RESIZE_CURSOR** -- Deprecated. replaced by Cursor.E_RESIZE_CURSOR.
- **static int HAND_CURSOR** -- Deprecated. replaced by Cursor.HAND_CURSOR.
- **static int ICONIFIED** -- This state bit indicates that frame is iconified.
- **static int MAXIMIZED_BOTH** -- This state bit mask indicates that frame is fully maximized (that is both horizontally and vertically).
- **static int MAXIMIZED_HORIZ** -- This state bit indicates that the frame is maximized in the horizontal direction.
- **static int MAXIMIZED_VERT** -- This state bit indicates that the frame is maximized in the vertical direction.
- **static int MOVE_CURSOR** -- Deprecated. replaced by Cursor.MOVE_CURSOR.
- **static int N_RESIZE_CURSOR** -- Deprecated. replaced by Cursor.N_RESIZE_CURSOR.
- **static int NE_RESIZE_CURSOR** -- Deprecated. replaced by Cursor.NE_RESIZE_CURSOR.
- **static int NORMAL** -- Frame is in the "normal" state.
- **static int NW_RESIZE_CURSOR** -- Deprecated. replaced by Cursor.NW_RESIZE_CURSOR.
- **static int S_RESIZE_CURSOR** -- Deprecated. replaced by Cursor.S_RESIZE_CURSOR.
- **static int SE_RESIZE_CURSOR** -- Deprecated. replaced by Cursor.SE_RESIZE_CURSOR.
- **static int SW_RESIZE_CURSOR** -- Deprecated. replaced by Cursor.SW_RESIZE_CURSOR.
- **static int TEXT_CURSOR** -- Deprecated. replaced by Cursor.TEXT_CURSOR.

- **static int W_RESIZE_CURSOR** -- Deprecated. replaced by Cursor.W_RESIZE_CURSOR.
- **static int WAIT_CURSOR** -- Deprecated. replaced by Cursor.WAIT_CURSOR.

Class Constructors

S.N.	Constructor & Description
1	**Frame()** Constructs a new instance of Frame that is initially invisible.
2	**Frame(GraphicsConfiguration gc)** Constructs a new, initially invisible Frame with the specified GraphicsConfiguration.
3	**Frame(String title)** Constructs a new, initially invisible Frame object with the specified title.
4	**Frame(String title, GraphicsConfiguration gc)** Constructs a new, initially invisible Frame object with the specified title and a GraphicsConfiguration.

Class Methods

S.N.	Method & Description
1	**void addNotify()** Makes this Frame displayable by connecting it to a native screen resource.
2	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Frame.
3	**int getCursorType()** Deprecated. As of JDK version 1.1, replaced by Component.getCursor().
4	**int getExtendedState()** Gets the state of this frame.
5	**static Frame[] getFrames()** Returns an array of all Frames created by this application.
6	**Image getIconImage()** Returns the image to be displayed as the icon for this frame.
7	**Rectangle getMaximizedBounds()** Gets maximized bounds for this frame.
8	**MenuBar getMenuBar()** Gets the menu bar for this frame.
9	**int getState()** Gets the state of this frame (obsolete).
10	**String getTitle()** Gets the title of the frame.
11	**boolean isResizable()** Indicates whether this frame is resizable by the user.
12	**boolean isUndecorated()** Indicates whether this frame is undecorated.
13	**protected String paramString()**

	Returns a string representing the state of this Frame.
14	**void remove(MenuComponent m)** Removes the specified menu bar from this frame.
15	**void removeNotify()** Makes this Frame undisplayable by removing its connection to its native screen resource.
16	**void setCursor(int cursorType)** Deprecated. As of JDK version 1.1, replaced by Component.setCursor(Cursor).
17	**void setExtendedState(int state)** Sets the state of this frame.
18	**void setIconImage(Image image)** Sets the image to be displayed as the icon for this window.
19	**void setMaximizedBounds(Rectangle bounds)** Sets the maximized bounds for this frame.
20	**void setMenuBar(MenuBar mb)** Sets the menu bar for this frame to the specified menu bar.
21	**void setResizable(boolean resizable)** Sets whether this frame is resizable by the user.
22	**void setState(int state)** Sets the state of this frame (obsolete).
23	**void setTitle(String title)** Sets the title for this frame to the specified string.
24	**void setUndecorated(boolean undecorated)** Disables or enables decorations for this frame.

Methods Inherited

This class inherits methods from the following classes:

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

Frame Example

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

AwtContainerDemo.java

```
package com.tutorialspoint.gui;


import java.awt.*;
```

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

public class AwtContainerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;
   private Label msglabel;

   public AwtContainerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtContainerDemo  awtContainerDemo = new AwtContainerDemo();
      awtContainerDemo.showFrameDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);
```

```
      msglabel = new Label();
      msglabel.setAlignment(Label.CENTER);
      msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");

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

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

   private void showFrameDemo(){
      headerLabel.setText("Container in action: Frame");

      final Frame frame = new Frame();
      frame.setSize(300, 300);
      frame.setLayout(new FlowLayout());
      frame.add(msglabel);
      frame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            frame.dispose();
         }
      });
      Button okButton = new Button("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);
   }
}
```

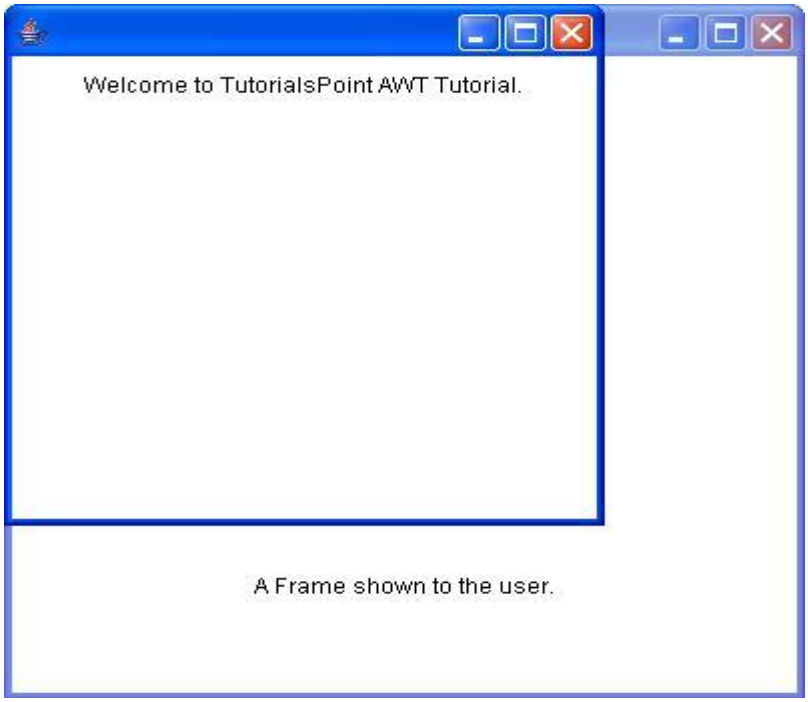
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT Window Class

The class **Window** is a top level window with no border and no menubar. It uses BorderLayout as default layout manager.

Class Declaration

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

```
public class Window
   extends Container
      implements Accessible
```

Class Constructors

S.N.	Constructor & Description
1	**Window(Frame owner)** Constructs a new, initially invisible window with the specified Frame as its owner.
2	**Window(Window owner)** Constructs a new, initially invisible window with the specified Window as its owner.
3	**Window(Window owner, GraphicsConfiguration gc)** Constructs a new, initially invisible window with the specified owner Window and a GraphicsConfiguration of a screen device.

Class Methods

S.N.	Method & Description
1	**void addNotify()** Makes this Window displayable by creating the connection to its native screen resource.
2	**void addPropertyChangeListener(PropertyChangeListener listener)** Adds a PropertyChangeListener to the listener list.
3	**void add Property ChangeListener(String property Name, Property Change Listener listener)** Adds a PropertyChangeListener to the listener list for a specific property.
4	**void addWindowFocusListener(WindowFocusListener l)** Adds the specified window focus listener to receive window events from this window.
5	**void addWindowListener(WindowListener l)** Adds the specified window listener to receive window events from this window.
6	**void addWindowStateListener(WindowStateListener l)** Adds the specified window state listener to receive window events from this window.
7	**void applyResourceBundle(ResourceBundle rb)** Deprecated. As of J2SE 1.4, replaced by Component.applyComponentOrientation.
8	**void applyResourceBundle(String rbName)** Deprecated. As of J2SE 1.4, replaced by Component.applyComponentOrientation.

9	**void createBufferStrategy(int numBuffers)** Creates a new strategy for multi-buffering on this component.
10	**void createBufferStrategy(int numBuffers, BufferCapabilities caps)** Creates a new strategy for multi-buffering on this component with the required buffer capabilities.
11	**void dispose()** Releases all of the native screen resources used by this Window, its subcomponents, and all of its owned children.
12	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Window.
13	**BufferStrategy getBufferStrategy()** Returns the BufferStrategy used by this component.
14	**boolean getFocusableWindowState()** Returns whether this Window can become the focused Window if it meets the other requirements outlined in FocusableWindow.
15	**Container getFocusCycleRootAncestor()** Always returns null because Windows have no ancestors; they represent the top of the Component hierarchy.
16	**Component getFocusOwner()** Returns the child Component of this Window that has focus if this Window is focused; returns null otherwise.
17	**Set<AWTKeyStroke> getFocusTraversalKeys(int id)** Gets a focus traversal key for this Window.
18	**GraphicsConfiguration getGraphicsConfiguration()** This method returns the GraphicsConfiguration used by this Window.
19	**List<Image> getIconImages()** Returns the sequence of images to be displayed as the icon for this window.
20	**InputContext getInputContext()** Gets the input context for this window.
21	**<T extends EventListener> T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this Window.
22	**Locale getLocale()** Gets the Locale object that is associated with this window, if the locale has been set.
23	**Dialog.ModalExclusionType getModalExclusionType()** Returns the modal exclusion type of this window.
24	**Component getMostRecentFocusOwner()** Returns the child Component of this Window that will receive the focus when this Window is focused.
25	**Window[] getOwnedWindows()** Return an array containing all the windows this window currently owns.
26	**Window getOwner()** Returns the owner of this window.
27	**static Window[] getOwnerlessWindows()**

	Returns an array of all Windows created by this application that have no owner.
28	**Toolkit getToolkit()** Returns the toolkit of this frame.
29	**String getWarningString()** Gets the warning string that is displayed with this window.
30	**WindowFocusListener[] getWindowFocusListeners()** Returns an array of all the window focus listeners registered on this window.
31	**WindowListener[] getWindowListeners()** Returns an array of all the window listeners registered on this window.
32	**static Window[] getWindows()** Returns an array of all Windows, both owned and ownerless, created by this application.
33	**WindowStateListener[] getWindowStateListeners()** Returns an array of all the window state listeners registered on this window.
34	**void hide()** Deprecated. As of JDK version 1.5, replaced by setVisible(boolean).
35	**boolean isActive()** Returns whether this Window is active.
36	**boolean isAlwaysOnTop()** Returns whether this window is an always-on-top window.
37	**boolean isAlwaysOnTopSupported()** Returns whether the always-on-top mode is supported for this window.
38	**boolean isFocusableWindow()** Returns whether this Window can become the focused Window, that is, whether this Window or any of its subcomponents can become the focus owner.
39	**boolean isFocusCycleRoot()** Always returns true because all Windows must be roots of a focus traversal cycle.
40	**boolean isFocused()** Returns whether this Window is focused.
41	**boolean isLocationByPlatform()** Returns true if this Window will appear at the default location for the native windowing system the next time this Window is made visible.
42	**boolean isShowing()** Checks if this Window is showing on screen.
43	**void pack()** Causes this Window to be sized to fit the preferred size and layouts of its subcomponents.
44	**void paint(Graphics g)** Paints the container.
45	**boolean postEvent(Event e)** Deprecated. As of JDK version 1.1 replaced by dispatchEvent(AWTEvent).
46	**protected void processEvent(AWTEvent e)** Processes events on this window.

47	**protected void processWindowEvent(WindowEvent e)** Processes window events occurring on this window by dispatching them to any registered WindowListener objects.
48	**protected void processWindowFocusEvent(WindowEvent e)** Processes window focus event occuring on this window by dispatching them to any registered WindowFocusListener objects.
49	**protected void processWindowStateEvent(WindowEvent e)** Processes window state event occuring on this window by dispatching them to any registered WindowStateListener objects.
50	**void removeNotify()** Makes this Container undisplayable by removing its connection to its native screen resource.
51	**void removeWindowFocusListener(WindowFocusListener l)** Removes the specified window focus listener so that it no longer receives window events from this window.
52	**void removeWindowListener(WindowListener l)** Removes the specified window listener so that it no longer receives window events from this window.
53	**void removeWindowStateListener(WindowStateListener l)** Removes the specified window state listener so that it no longer receives window events from this window.
54	**void reshape(int x, int y, int width, int height)** Deprecated. As of JDK version 1.1, replaced by setBounds(int, int, int, int).
55	**void setAlwaysOnTop(boolean alwaysOnTop)** Sets whether this window should always be above other windows.
56	**void setBounds(int x, int y, int width, int height)** Moves and resizes this component.
57	**void setBounds(Rectangle r)** Moves and resizes this component to conform to the new bounding rectangle r.
58	**void setCursor(Cursor cursor)** Set the cursor image to a specified cursor.
59	**void setFocusableWindowState(boolean focusableWindowState)** Sets whether this Window can become the focused Window if it meets the other requirements outlined in isFocusableWindow.
60	**void setFocusCycleRoot(boolean focusCycleRoot)** Does nothing because Windows must always be roots of a focus traversal cycle.
61	**void setIconImage(Image image)** Sets the image to be displayed as the icon for this window.
62	**void setIconImages(List<? extends Image> icons)** Sets the sequence of images to be displayed as the icon for this window.
63	**void setLocationByPlatform(boolean locationByPlatform)** Sets whether this Window should appear at the default location for the native windowing system or at the current location (returned by getLocation) the next time the Window is made visible.
64	**void setLocationRelativeTo(Component c)** Sets the location of the window relative to the specified component.

65	**void setMinimumSize(Dimension minimumSize)** Sets the minimum size of this window to a constant value.
66	**void setModalExclusionType(Dialog.ModalExclusionType exclusionType)** Specifies the modal exclusion type for this window.
67	**void setSize(Dimension d)** Resizes this component so that it has width d.width and height d.height.
68	**void setSize(int width, int height)** Resizes this component so that it has width width and height height.
69	**void setVisible(boolean b)** Shows or hides this Window depending on the value of parameter b.
70	**void show()** Deprecated. As of JDK version 1.5, replaced by setVisible(boolean).
71	**void toBack()** If this Window is visible, sends this Window to the back and may cause it to lose focus or activation if it is the focused or active Window.
72	**void toFront()** If this Window is visible, brings this Window to the front and may make it the focused Window.

Methods Inherited

This class inherits methods from the following classes:

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

Window Example

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

AwtContainerDemo.java

```
package com.tutorialspoint.gui;

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

public class AwtContainerDemo {
   private Frame mainFrame;
   private Label headerLabel;
```

```
   private Label statusLabel;
   private Panel controlPanel;
   private Label msglabel;

   public AwtContainerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtContainerDemo  awtContainerDemo = new AwtContainerDemo();
      awtContainerDemo.showFrameDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      msglabel = new Label();
      msglabel.setAlignment(Label.CENTER);
      msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");

      controlPanel = new Panel();
```

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

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

   private void showWindowDemo(){
      headerLabel.setText("Container in action: Window");
      final MessageWindow window =
         new MessageWindow(mainFrame,
         "Welcome to TutorialsPoint AWT Tutorial.");

      Button okButton = new Button("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 Window{
      private String message;

      public MessageWindow(Frame parent, String message) {
         super(parent);
         this.message = message;
         setSize(300, 300);
         setLocationRelativeTo(parent);
```

```
         setBackground(Color.gray);
      }

      public void paint(Graphics g) {
         super.paint(g);
         g.drawRect(0,0,getSize().width - 1,getSize().height - 1);
         g.drawString(message,50,150);
      }
   }
}
```

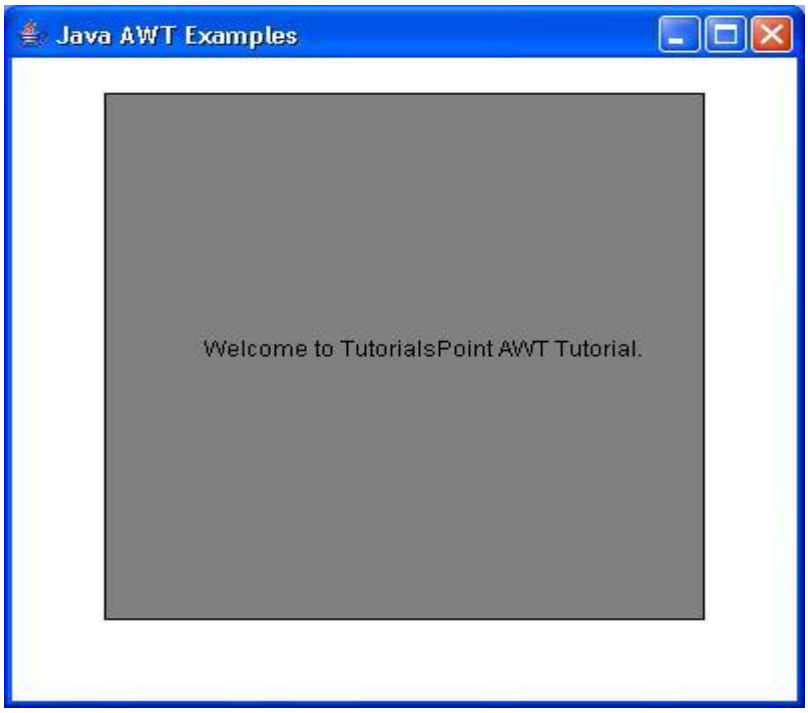
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

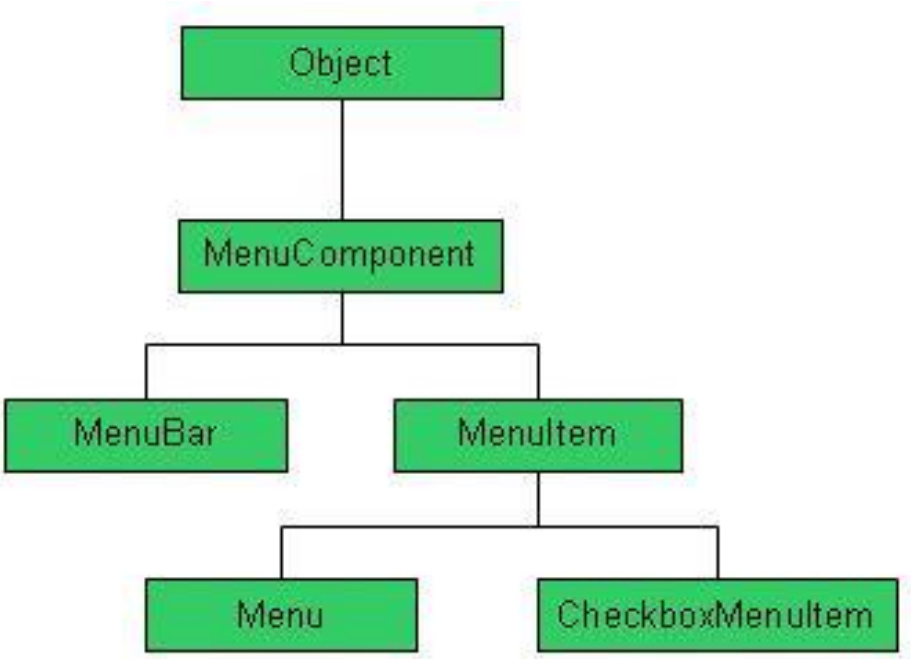
Output



10. AWT – 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 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	**MenuComponent** It is the top level class for all menu related controls.
2	**MenuBar** The MenuBar object is associated with the top-level window.
3	**MenuItem** The items in the menu must belong to the MenuItem or any of its subclass.
4	**Menu**

	The Menu object is a pull-down menu component, which is displayed from the menu bar.
5	**CheckboxMenuItem** CheckboxMenuItem is subclass of MenuItem.
6	**PopupMenu** PopupMenu can be dynamically popped up at a specified position within a component.

AWT MenuComponent Class

MenuComponent is an abstract class and is the superclass for all menu-related components.

Class Declaration

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

```
public abstract class MenuComponent
   extends Object
      implements Serializable
```

Class Constructors

S.N.	Constructor & Description
1	**MenuComponent()** Creates a MenuComponent.

Class Methods

Void dispatchEvent(AWTEvent e)

S.N.	Method & Description
1	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this MenuComponent.
2	**Font getFont()** Gets the font used for this menu component.
3	**String getName()** Gets the name of the menu component.
4	**MenuContainer getParent()** Returns the parent container for this menu component.
5	**java.awt.peer.MenuComponentPeer getPeer()** Deprecated. As of JDK version 1.1, programs should not directly manipulate peers.
6	**protected Object getTreeLock()** Gets this component's locking object (the object that owns the thread sychronization monitor) for AWT component-tree and layout operations.

7	**protected String paramString()** Returns a string representing the state of this MenuComponent.
8	**boolean postEvent(Event evt)** Deprecated. As of JDK version 1.1, replaced by dispatchEvent.
9	**protected void processEvent(AWTEvent e)** Processes events occurring on this menu component.
10	**void removeNotify()** Removes the menu component's peer.
11	**void setFont(Font f)** Sets the font to be used for this menu component to the specified font.
12	**void setName(String name)** Sets the name of the component to the specified string.
13	**String toString()** Returns a representation of this menu component as a string.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

AWT MenuBar Class

The MenuBar class provides menu bar bound to a frame and is platform specific.

Class Declaration

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

```
public class MenuBar
   extends MenuComponent
      implements MenuContainer, Accessible
```

Class Constructors

S.N.	Constructor & Description
1	**MenuBar()** Creates a new menu bar.

Class Methods

S.N.	Method & Description
1	**void dispatchEvent(AWTEvent e)**
2	**Menu add(Menu m)** Adds the specified menu to the menu bar.
3	**void addNotify()** Creates the menu bar's peer.

4	**int countMenus()** Deprecated. As of JDK version 1.1, replaced by getMenuCount().
5	**void deleteShortcut(MenuShortcut s)** Deletes the specified menu shortcut.
6	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this MenuBar.
7	**Menu getHelpMenu()** Gets the help menu on the menu bar.
8	**Menu getMenu(int i)** Gets the specified menu.
9	**int getMenuCount()** Gets the number of menus on the menu bar.
10	**MenuItem getShortcutMenuItem(MenuShortcut s)** Gets the instance of MenuItem associated with the specified MenuShortcut object, or null if none of the menu items being managed by this menu bar is associated with the specified menu shortcut.
11	**void remove(int index)** Removes the menu located at the specified index from this menu bar.
12	**void remove(MenuComponent m)** Removes the specified menu component from this menu bar.
13	**void removeNotify()** Removes the menu bar's peer.
14	**void setHelpMenu(Menu m)** Sets the specified menu to be this menu bar's help menu.
15	**Enumeration shortcuts()** Gets an enumeration of all menu shortcuts this menu bar is managing.

Methods Inherited

This class inherits methods from the following classes:

- java.awt.MenuComponent
- java.lang.Object

MenuBar Example

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

AWTMenuDemo.java

```
package com.tutorialspoint.gui;


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

```
public class AWTMenuDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AWTMenuDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AWTMenuDemo  awtMenuDemo = new AWTMenuDemo();
      awtMenuDemo.showMenuDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

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

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

   private void showMenuDemo(){
      //create a menu bar
      final MenuBar menuBar = new MenuBar();

      //create menus
      Menu fileMenu = new Menu("File");
      Menu editMenu = new Menu("Edit");
      final Menu aboutMenu = new Menu("About");

      //create menu items
      MenuItem newMenuItem =
         new MenuItem("New",new MenuShortcut(KeyEvent.VK_N));
      newMenuItem.setActionCommand("New");

      MenuItem openMenuItem = new MenuItem("Open");
      openMenuItem.setActionCommand("Open");

      MenuItem saveMenuItem = new MenuItem("Save");
      saveMenuItem.setActionCommand("Save");

      MenuItem exitMenuItem = new MenuItem("Exit");
      exitMenuItem.setActionCommand("Exit");

      MenuItem cutMenuItem = new MenuItem("Cut");
      cutMenuItem.setActionCommand("Cut");

      MenuItem copyMenuItem = new MenuItem("Copy");
```

```
      copyMenuItem.setActionCommand("Copy");

      MenuItem pasteMenuItem = new MenuItem("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 CheckboxMenuItem showWindowMenu =
         new CheckboxMenuItem("Show About", true);
      showWindowMenu.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
            if(showWindowMenu.getState()){
               menuBar.add(aboutMenu);
            }else{
               menuBar.remove(aboutMenu);
            }
         }
      });

      //add menu items to menus
      fileMenu.add(newMenuItem);
      fileMenu.add(openMenuItem);
      fileMenu.add(saveMenuItem);
      fileMenu.addSeparator();
      fileMenu.add(showWindowMenu);
```

```
      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);

      //add menubar to the frame
      mainFrame.setMenuBar(menuBar);
      mainFrame.setVisible(true);
   }

   class MenuItemListener implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         statusLabel.setText(e.getActionCommand()
            + " MenuItem clicked.");
      }
   }
}
```

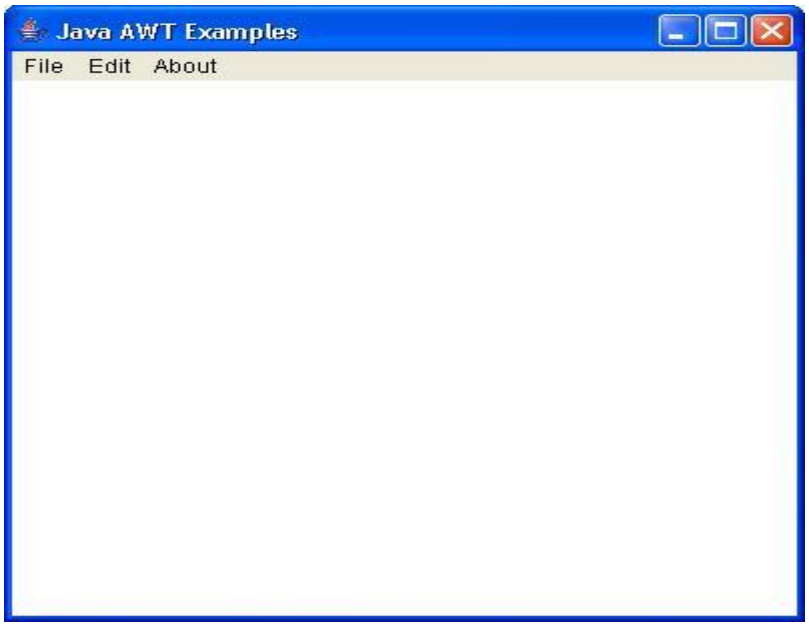
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT MenuItem Class

The MenuBar class represents the actual item in a menu. All items in a menu should derive from class MenuItem, or one of its subclasses. By default, it embodies a simple labeled menu item.

Class Declaration

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

```
public class MenuItem
   extends MenuComponent
      implements Accessible
```

Class Constructors

S.N.	Constructor & Description
1	**MenuItem()** Constructs a new MenuItem with an empty label and no keyboard shortcut.
2	**MenuItem(String label)** Constructs a new MenuItem with the specified label and no keyboard shortcut.
3	**MenuItem(String label, MenuShortcut s)** Create a menu item with an associated keyboard shortcut.

Class Methods

S.N.	Method & Description
1	**void addActionListener(ActionListener l)** Adds the specified action listener to receive action events from this menu item.
2	**void addNotify()** Creates the menu item's peer.
3	**void deleteShortcut()** Delete any MenuShortcut object associated with this menu item.
4	**void disable()** Deprecated. As of JDK version 1.1, replaced by setEnabled(boolean).
5	**protected void disableEvents(long eventsToDisable)** Disables event delivery to this menu item for events defined by the specified event mask parameter.
6	**void enable()** Deprecated. As of JDK version 1.1, replaced by setEnabled(boolean).
7	**void enable(boolean b)** Deprecated. As of JDK version 1.1, replaced by setEnabled(boolean).
8	**protected void enableEvents(long eventsToEnable)** Enables event delivery to this menu item for events to be defined by the specified event mask parameter.
9	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this MenuItem.
10	**String getActionCommand()** Gets the command name of the action event that is fired by this menu item.
11	**ActionListener[] getActionListeners()** Returns an array of all the action listeners registered on this menu item.
12	**String getLabel()** Gets the label for this menu item.
13	**EventListener[] getListeners(Class listenerType)** Returns an array of all the objects currently registered as FooListeners upon this MenuItem.
14	**MenuShortcut getShortcut()** Get the MenuShortcut object associated with this menu item.
15	**boolean isEnabled()** Checks whether this menu item is enabled.
16	**String paramString()** Returns a string representing the state of this MenuItem.
17	**protected void processActionEvent(ActionEvent e)** Processes action events occurring on this menu item, by dispatching them to any registered ActionListener objects.
18	**protected void processEvent(AWTEvent e)** Processes events on this menu item.
19	**void removeActionListener(ActionListener l)** Removes the specified action listener so it no longer receives action events from this menu item.
20	**void setActionCommand(String command)**

	Sets the command name of the action event that is fired by this menu item.
21	**void setEnabled(boolean b)** Sets whether or not this menu item can be chosen.
22	**void setLabel(String label)** Sets the label for this menu item to the specified label.
23	**void setShortcut(MenuShortcut s)** Set the MenuShortcut object associated with this menu item.

Methods Inherited

This class inherits methods from the following classes:

- java.awt.MenuComponent
- java.lang.Object

MenuItem Example

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

AWTMenuDemo.java

```
package com.tutorialspoint.gui;

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

public class AWTMenuDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AWTMenuDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AWTMenuDemo  awtMenuDemo = new AWTMenuDemo();
```

```
      awtMenuDemo.showMenuDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

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

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

   private void showMenuDemo(){
      //create a menu bar
      final MenuBar menuBar = new MenuBar();

      //create menus
      Menu fileMenu = new Menu("File");
```

```
      Menu editMenu = new Menu("Edit");
      final Menu aboutMenu = new Menu("About");

      //create menu items
      MenuItem newMenuItem =
         new MenuItem("New",new MenuShortcut(KeyEvent.VK_N));
      newMenuItem.setActionCommand("New");

      MenuItem openMenuItem = new MenuItem("Open");
      openMenuItem.setActionCommand("Open");

      MenuItem saveMenuItem = new MenuItem("Save");
      saveMenuItem.setActionCommand("Save");

      MenuItem exitMenuItem = new MenuItem("Exit");
      exitMenuItem.setActionCommand("Exit");

      MenuItem cutMenuItem = new MenuItem("Cut");
      cutMenuItem.setActionCommand("Cut");

      MenuItem copyMenuItem = new MenuItem("Copy");
      copyMenuItem.setActionCommand("Copy");

      MenuItem pasteMenuItem = new MenuItem("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 CheckboxMenuItem showWindowMenu =
         new CheckboxMenuItem("Show About", true);
      showWindowMenu.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
            if(showWindowMenu.getState()){
               menuBar.add(aboutMenu);
            }else{
               menuBar.remove(aboutMenu);
            }
         }
      });

      //add menu items to menus
      fileMenu.add(newMenuItem);
      fileMenu.add(openMenuItem);
      fileMenu.add(saveMenuItem);
      fileMenu.addSeparator();
      fileMenu.add(showWindowMenu);
      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);
```

```
      //add menubar to the frame
      mainFrame.setMenuBar(menuBar);
      mainFrame.setVisible(true);
}

   class MenuItemListener implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         statusLabel.setText(e.getActionCommand()
            + " MenuItem clicked.");
      }
   }
}
```

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

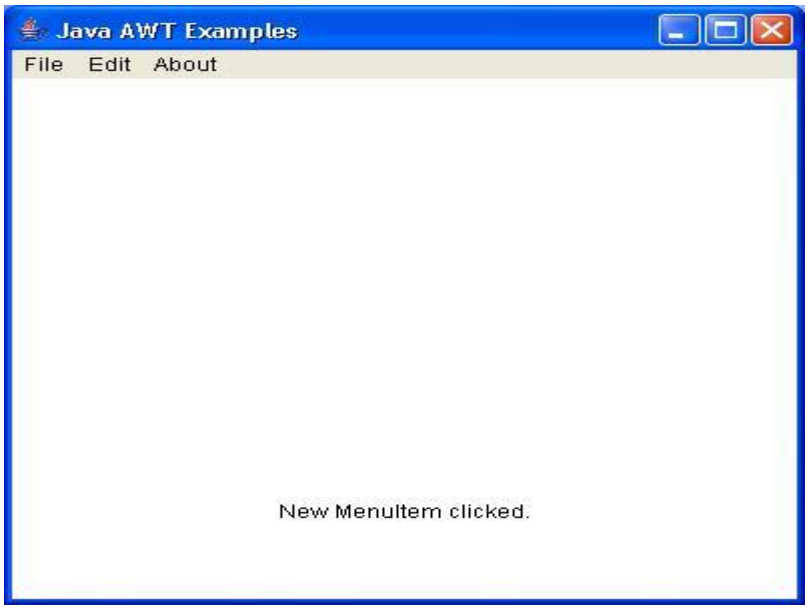
```
D:\AWT>javac com\tutorialspoint\gui\AWTMenuDemo.java
```

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

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

Output

(Click on File Menu. Select any menu item.)



AWT Menu Class

The Menu class represents pull-down menu component, which is deployed from a menu bar.

Class Declaration

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

```
public class Menu
   extends MenuItem
      implements MenuContainer, Accessible
```

Class Constructors

S.N.	Constructor & Description
1	**Menu()** Constructs a new menu with an empty label.
2	**Menu(String label)** Constructs a new menu with the specified label.
3	**Menu(String label, boolean tearOff)** Constructs a new menu with the specified label, indicating whether the menu can be torn off.

Class Methods

S.N.	Method & Description
1	**MenuItem add(MenuItem mi)** Adds the specified menu item to this menu.
2	**void add(String label)** Adds an item with the specified label to this menu.
3	**void addNotify()** Creates the menu's peer.
4	**void addSeparator()** Adds a separator line, or a hypen, to the menu at the current position.
5	**int countItems()** Deprecated. As of JDK version 1.1, replaced by getItemCount().
6	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Menu.
7	**MenuItem getItem(int index)** Gets the item located at the specified index of this menu.
8	**int getItemCount()** Get the number of items in this menu.
9	**void insert(MenuItem menuitem, int index)** Inserts a menu item into this menu at the specified position.

10	**void insert(String label, int index)** Inserts a menu item with the specified label into this menu at the specified position.
11	**void insertSeparator(int index)** Inserts a separator at the specified position.
12	**boolean isTearOff()** Indicates whether this menu is a tear-off menu.
13	**String paramString()** Returns a string representing the state of this Menu.
14	**void remove(int index)** Removes the menu item at the specified index from this menu.
15	**void remove(MenuComponent item)** Removes the specified menu item from this menu.
16	**void removeAll()** Removes all items from this menu.
17	**void removeNotify()** Removes the menu's peer.

Methods Inherited

This class inherits methods from the following classes:

- java.awt.MenuItem
- java.awt.MenuComponent
- java.lang.Object

Menu Example

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

AWTMenuDemo.java

```
package com.tutorialspoint.gui;


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


public class AWTMenuDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;
```

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

public static void main(String[] args){
   AWTMenuDemo  awtMenuDemo = new AWTMenuDemo();
   awtMenuDemo.showMenuDemo();
}

private void prepareGUI(){
   mainFrame = new Frame("Java AWT 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 Label();
   headerLabel.setAlignment(Label.CENTER);
   statusLabel = new Label();
   statusLabel.setAlignment(Label.CENTER);
   statusLabel.setSize(350,100);

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

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

```
private void showMenuDemo(){
   //create a menu bar
   final MenuBar menuBar = new MenuBar();

   //create menus
   Menu fileMenu = new Menu("File");
   Menu editMenu = new Menu("Edit");
   final Menu aboutMenu = new Menu("About");

   //create menu items
   MenuItem newMenuItem =
      new MenuItem("New",new MenuShortcut(KeyEvent.VK_N));
   newMenuItem.setActionCommand("New");

   MenuItem openMenuItem = new MenuItem("Open");
   openMenuItem.setActionCommand("Open");

   MenuItem saveMenuItem = new MenuItem("Save");
   saveMenuItem.setActionCommand("Save");

   MenuItem exitMenuItem = new MenuItem("Exit");
   exitMenuItem.setActionCommand("Exit");

   MenuItem cutMenuItem = new MenuItem("Cut");
   cutMenuItem.setActionCommand("Cut");

   MenuItem copyMenuItem = new MenuItem("Copy");
   copyMenuItem.setActionCommand("Copy");

   MenuItem pasteMenuItem = new MenuItem("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 CheckboxMenuItem showWindowMenu =
         new CheckboxMenuItem("Show About", true);
      showWindowMenu.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
            if(showWindowMenu.getState()){
               menuBar.add(aboutMenu);
            }else{
               menuBar.remove(aboutMenu);
            }
         }
      });

      //add menu items to menus
      fileMenu.add(newMenuItem);
      fileMenu.add(openMenuItem);
      fileMenu.add(saveMenuItem);
      fileMenu.addSeparator();
      fileMenu.add(showWindowMenu);
      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);

      //add menubar to the frame
      mainFrame.setMenuBar(menuBar);
      mainFrame.setVisible(true);
   }

   class MenuItemListener implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         statusLabel.setText(e.getActionCommand()
            + " MenuItem clicked.");
      }
   }
}
```

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

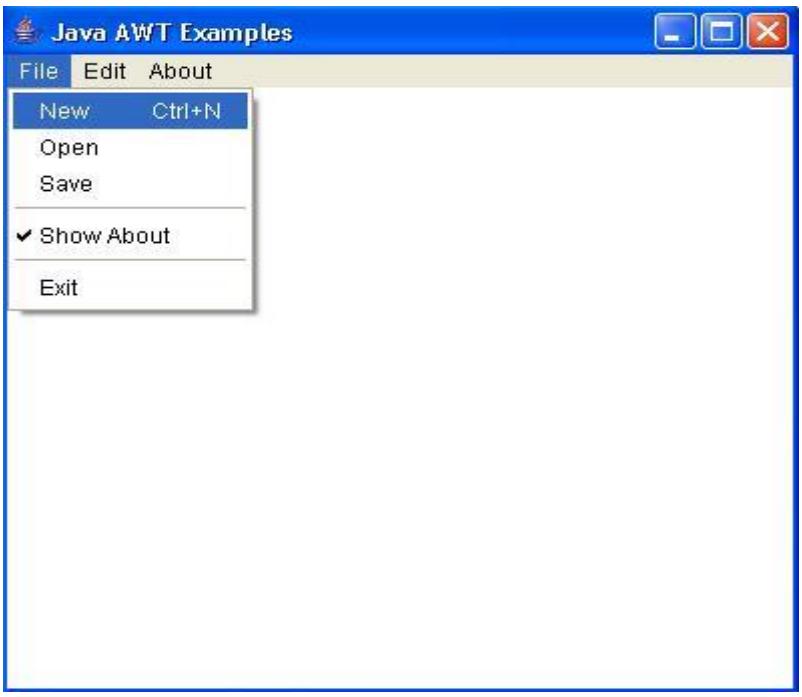
```
D:\AWT>javac com\tutorialspoint\gui\AWTMenuDemo.java
```

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

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

Output

(Click on File Menu):



AWT CheckboxMenuItem Class

The CheckboxMenuItem class represents a check box, which can be included in a menu. Selecting the check box in the menu changes the control's state from **on** to **off** or from **off** to **on**.

Class Declaration

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

```
public class CheckboxMenuItem
   extends MenuItem
      implements ItemSelectable, Accessible
```

Class Constructors

S.N.	Constructor & Description
1	**CheckboxMenuItem()** Create a check box menu item with an empty label.
2	**CheckboxMenuItem(label)** Create a check box menu item with the specified label.
3	**CheckboxMenuItem(label, boolean state)** Create a check box menu item with the specified label and state.

Class Methods

S.N.	Method & Description
1	**void addItemListener(ItemListener l)** Adds the specified item listener to receive item events from this check box menu item.
2	**void addNotify()** Creates the peer of the checkbox item.
3	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this CheckboxMenuItem.
4	**ItemListener[] getItemListeners()** Returns an array of all the item listeners registered on this checkbox menuitem.
5	**<T extends EventListener> T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this CheckboxMenuItem.
6	**Object[] getSelectedObjects()** Returns an array (length 1) containing the checkbox menu item label or null if the checkbox is not selected.
7	**boolean getState()** Determines whether the state of this check box menu item is "on" or "off."
8	**param()** Returns a representing the state of this CheckBoxMenuItem.
9	**protected void processEvent(AWTEvent e)** Processes events on this check box menu item.
10	**protected void processItemEvent(ItemEvent e)** Processes item events occurring on this check box menu item by dispatching them to any registered ItemListener objects.
11	**void removeItemListener(ItemListener l)** Removes the specified item listener so that it no longer receives item events from this check box menu item.
12	**void setState(boolean b)** Sets this check box menu item to the specifed state.

Methods Inherited

This class inherits methods from the following classes:

- java.awt.MenuItem
- java.awt.MenuComponent
- java.lang.Object

CheckboxMenuItem Example

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

AWTMenuDemo.java

```
package com.tutorialspoint.gui;

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

public class AWTMenuDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AWTMenuDemo(){
      prepareGUI();
   }

   public static void main(
[] args){
      AWTMenuDemo  awtMenuDemo = new AWTMenuDemo();
      awtMenuDemo.showMenuDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
```

```
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

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

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

   private void showMenuDemo(){
      //create a menu bar
      final MenuBar menuBar = new MenuBar();

      //create menus
      Menu fileMenu = new Menu("File");
      Menu editMenu = new Menu("Edit");
      final Menu aboutMenu = new Menu("About");

      //create menu items
      MenuItem newMenuItem =
         new MenuItem("New",new MenuShortcut(KeyEvent.VK_N));
      newMenuItem.setActionCommand("New");

      MenuItem openMenuItem = new MenuItem("Open");
      openMenuItem.setActionCommand("Open");

      MenuItem saveMenuItem = new MenuItem("Save");
      saveMenuItem.setActionCommand("Save");
```

```
MenuItem exitMenuItem = new MenuItem("Exit");
exitMenuItem.setActionCommand("Exit");

MenuItem cutMenuItem = new MenuItem("Cut");
cutMenuItem.setActionCommand("Cut");

MenuItem copyMenuItem = new MenuItem("Copy");
copyMenuItem.setActionCommand("Copy");

MenuItem pasteMenuItem = new MenuItem("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 CheckboxMenuItem showWindowMenu =
   new CheckboxMenuItem("Show About", true);
showWindowMenu.addItemListener(new ItemListener() {
   public void itemStateChanged(ItemEvent e) {
      if(showWindowMenu.getState()){
         menuBar.add(aboutMenu);
      }else{
         menuBar.remove(aboutMenu);
      }
   }
```

```
      });

      //add menu items to menus
      fileMenu.add(newMenuItem);
      fileMenu.add(openMenuItem);
      fileMenu.add(saveMenuItem);
      fileMenu.addSeparator();
      fileMenu.add(showWindowMenu);
      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);

      //add menubar to the frame
      mainFrame.setMenuBar(menuBar);
      mainFrame.setVisible(true);
   }

   class MenuItemListener implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         statusLabel.setText(e.getActionCommand()
            + " MenuItem clicked.");
      }
   }
}
```

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

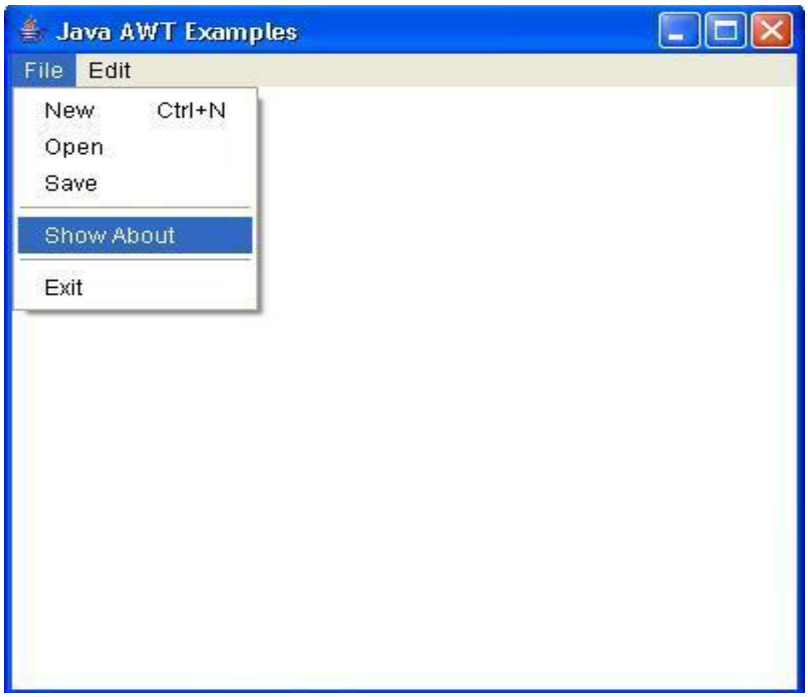
```
D:\AWT>javac com\tutorialspoint\gui\AWTMenuDemo.java
```

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

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

Output

(Click on File Menu. Unselect "Show About" menu item):



AWT PopupMenu Class

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 **java.awt.PopupMenu** class:

```
public class CheckboxMenuItem
   extends MenuItem
      implements ItemSelectable, Accessible
```

Class Constructors

S.N.	Constructor & Description
1	**PopupMenu()** Creates a new popup menu with an empty name.
2	**PopupMenu(String label)** Creates a new popup menu with the specified name.

Class Methods

S.N.	Method & Description
1	**void addNotify()** Creates the popup menu's peer.
2	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this PopupMenu.
3	**MenuContainer getParent()** Returns the parent container for this menu component.
4	**void show(Component origin, int x, int y)** Shows the popup menu at the x, y position relative to an origin component.

Methods Inherited

This class inherits methods from the following classes:

- java.awt.MenuItem
- java.awt.MenuComponent
- java.lang.Object

PopupMenu Example

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

AWTMenuDemo.java

```
package com.tutorialspoint.gui;

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

public class AWTMenuDemo {
   private Frame mainFrame;
   private Label headerLabel;
```

```
   private Label statusLabel;
   private Panel controlPanel;

   public AWTMenuDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AWTMenuDemo  awtMenuDemo = new AWTMenuDemo();
      awtMenuDemo.showPopupMenuDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT 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 Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

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

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

```
      mainFrame.setVisible(true);
   }

   private void showPopupMenuDemo(){
      final PopupMenu editMenu = new PopupMenu("Edit");

      MenuItem cutMenuItem = new MenuItem("Cut");
      cutMenuItem.setActionCommand("Cut");

      MenuItem copyMenuItem = new MenuItem("Copy");
      copyMenuItem.setActionCommand("Copy");

      MenuItem pasteMenuItem = new MenuItem("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);

      controlPanel.addMouseListener(new MouseAdapter() {
         public void mouseClicked(MouseEvent e) {
               editMenu.show(controlPanel, e.getX(), e.getY());
         }
      });
      controlPanel.add(editMenu);

      mainFrame.setVisible(true);
```

```
   }

   class MenuItemListener implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         statusLabel.setText(e.getActionCommand()
            + " MenuItem clicked.");
      }
   }
}
```

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

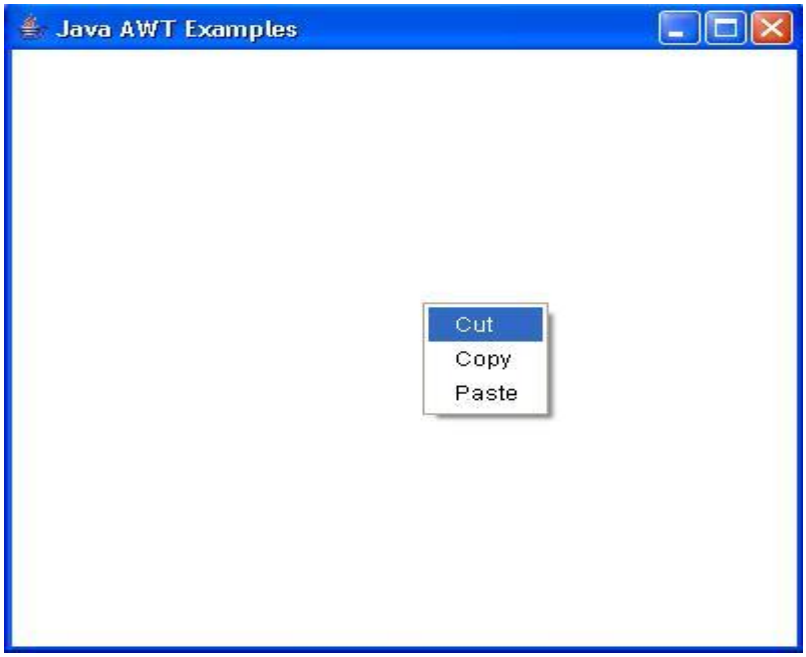
```
D:\AWT>javac com\tutorialspoint\gui\AWTMenuDemo.java
```

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

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

Output

(Click in the middle on the screen):



11. AWT – GRAPHICS CLASSES

Graphic controls allow application to draw onto the component or on image.

Graphics Controls

Sr. No.	Control & Description
1	**Graphics** It is the top level abstract class for all graphics contexts.
2	**Graphics2D** It is a subclass of Graphics class and provides more sophisticated control over geometry, coordinate transformations, color management, and text layout.
3	**Arc2D** Arc2D is the abstract superclass for all objects, which stores a 2D arc, defined by a framing rectangle, start angle, angular extent (length of the arc), and a closure type (OPEN, CHORD, or PIE).
4	**CubicCurve2D** The CubicCurve2D class is the abstract superclass fpr all objects, which stores a 2D cubic curve segment and it defines a cubic parametric curve segment in (x & y) coordinate space.
5	**Ellipse2D** The Ellipse2D is the abstract superclass for all objects, which stores a 2D ellipse and it describes an ellipse that is defined by a framing rectangle.
6	**Rectangle2D** The Rectangle2D class is an abstract superclass for all objects, which stores a 2D rectangle and it describes a rectangle defined by a location (x & y) and dimension (w x h).
7	**QuadCurve2D** The QuadCurve2D class is an abstract superclass for all objects, which stores a 2D quadratic curve segment and it describes a quadratic parametric curve segment in (x & y) coordinate space.
8	**Line2D** This Line2D represents a line segment in (x & y) coordinate space.
9	**Font** The Font class represents fonts, which are used to render text in a visible way.

10	**Color** The Color class is used encapsulate colors in the default sRGB color space or colors in arbitrary color spaces, identified by a ColorSpace.
11	**BasicStroke** The BasicStroke class defines a basic set of rendering attributes for the outlines of graphics primitives, which are rendered with a Graphics2D object. It has its Stroke attribute set to this BasicStroke.

AWT Graphics Class

The Graphics class is the abstract super class for all graphics contexts, which allows an application to draw onto components that can be realized on various devices, or onto off-screen images as well.

A Graphics object encapsulates all state information required for the basic rendering operations that Java supports. State information includes the following properties.

- The Component object on which to draw.
- A translation origin for rendering and clipping coordinates.
- The current clip.
- The current color.
- The current font.
- The current logical pixel operation function.
- The current XOR alternation color

Class Declaration

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

```
public abstract class Graphics
   extends Object
```

Class Constructors

S.N.	Constructor & Description
1	**Graphics()** Constructs a new Graphics object.

Class Methods

S.N.	Method & Description
1	**abstract void clearRect(int x, int y, int width, int height)** Clears the specified rectangle by filling it with the background color of the current drawing surface.
2	**abstract void clipRect(int x, int y, int width, int height)** Intersects the current clip with the specified rectangle.
3	**abstract void copyArea(int x, int y, int width, int height, int dx, int dy)** Copies an area of the component by a distance specified by dx and dy.
4	**abstract Graphics create()** Creates a new Graphics object that is a copy of this Graphics object.
5	**Graphics create(int x, int y, int width, int height)** Creates a new Graphics object based on this Graphics object, but with a new translation and clip area.
6	**abstract void dispose()** Disposes of this graphics context and releases any system resources, which it is using.
7	**void draw3DRect(int x, int y, int width, int height, boolean raised)** Draws a 3-D highlighted outline of the specified rectangle.
8	**abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)** Draws the outline of a circular or elliptical arc covering the specified rectangle.
9	**void drawBytes(byte[] data, int offset, int length, int x, int y)** Draws the text given by the specified byte array, using this graphics context's current font and color.
10	**void drawChars(char[] data, int offset, int length, int x, int y)** Draws the text given by the specified character array, using this graphics context's current font and color.
11	**abstract boolean drawImage(Image img, int x, int y, Color bgcolor, ImageObserver observer)** Draws as much of the specified image as is currently available.
12	**abstract boolean drawImage(Image img, int x, int y, ImageObserver observer)** Draws as much of the specified image as is currently available.
13	**abstract boolean drawImage(Image img, int x, int y, int width, int height, Color bgcolor, ImageObserver observer)** Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.
14	**abstract boolean drawImage(Image img, int x, int y, int width, int height, ImageObserver observer)** Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.
15	**abstract boolean drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, Color bgcolor, ImageObserver observer)**

	Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface.
16	**abstract boolean drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, ImageObserver observer)** Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface.
17	**abstract void drawLine(int x1, int y1, int x2, int y2)** Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.
18	**abstract void drawOval(int x, int y, int width, int height)** Draws the outline of an oval.
19	**abstract void drawPolygon(int[] xPoints, int[] yPoints, int nPoints)** Draws a closed polygon defined by arrays of x and y coordinates.
20	**void drawPolygon(Polygon p)** Draws the outline of a polygon defined by the specified Polygon object.
21	**abstract void drawPolyline(int[] xPoints, int[] yPoints, int nPoints)** Draws a sequence of connected lines defined by arrays of x and y coordinates.
22	**void drawRect(int x, int y, int width, int height)** Draws the outline of the specified rectangle.
23	**abstract void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)** Draws an outlined round-cornered rectangle using this graphics context's current color.
24	**abstract void drawString(AttributedCharacterIterator iterator, int x, int y)** Renders the text of the specified iterator applying its attributes in accordance with the specification of the TextAttribute class.
25	**abstract void drawString(String str, int x, int y)** Draws the text given by the specified string, using this graphics context's current font and color.
26	**void fill3DRect(int x, int y, int width, int height, boolean raised)** Paints a 3-D highlighted rectangle filled with the current color.
27	**abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)** Fills a circular or elliptical arc covering the specified rectangle.
28	**abstract void fillOval(int x, int y, int width, int height)** Fills an oval bounded by the specified rectangle with the current color.
29	**abstract void fillPolygon(int[] xPoints, int[] yPoints, int nPoints)** Fills a closed polygon defined by arrays of x and y coordinates.
30	**void fillPolygon(Polygon p)** Fills the polygon defined by the specified Polygon object with the graphics context's current color.
31	**abstract void fillRect(int x, int y, int width, int height)**

	Fills the specified rectangle.
32	**abstract void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)** Fills the specified rounded corner rectangle with the current color.
33	**void finalize()** Disposes of this graphics context once it is no longer referenced.
34	**abstract Shape getClip()** Gets the current clipping area.
35	**abstract Rectangle getClipBounds()** Returns the bounding rectangle of the current clipping area.
36	**Rectangle getClipBounds(Rectangle r)** Returns the bounding rectangle of the current clipping area.
37	**Rectangle getClipRect()** Deprecated. As of JDK version 1.1, replaced by getClipBounds().
38	**abstract Color getColor()** Gets this graphics context's current color.
39	**abstract Font getFont()** Gets the current font.
40	**FontMetrics getFontMetrics()** Gets the font metrics of the current font.
41	**abstract FontMetrics getFontMetrics(Font f)** Gets the font metrics for the specified font.
42	**boolean hitClip(int x, int y, int width, int height)** Returns true if the specified rectangular area might intersect the current clipping area.
43	**abstract void setClip(int x, int y, int width, int height)** Sets the current clip to the rectangle specified by the given coordinates.
44	**abstract void setClip(Shape clip)** Sets the current clipping area to an arbitrary clip shape.
45	**abstract void setColor(Color c)** Sets this graphics context's current color to the specified color.
46	**abstract void setFont(Font font)** Sets this graphics context's font to the specified font.
47	**abstract void setPaintMode()** Sets the paint mode of this graphics context to overwrite the destination with this graphics context's current color.
48	**abstract void setXORMode(Color c1)** Sets the paint mode of this graphics context to alternate between this graphics context's current color and the new specified color.
49	**String toString()** Returns a String object representing this Graphics object's value.
50	**abstract void translate(int x, int y)** Translates the origin of the graphics context to the point (x & y) in the current coordinate system.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

Graphics Example

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

AWTGraphicsDemo.java

```
package com.tutorialspoint.gui;

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class AWTGraphicsDemo extends Frame {

   public AWTGraphicsDemo(){
      super("Java AWT Examples");
      prepareGUI();
   }

   public static void main(String[] args){
      AWTGraphicsDemo  awtGraphicsDemo = new AWTGraphicsDemo();
      awtGraphicsDemo.setVisible(true);
   }

   private void prepareGUI(){
      setSize(400,400);
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
```

```
   }

   @Override
   public void paint(Graphics g) {
      g.setColor(Color.GRAY);
      Font font = new Font("Serif", Font.PLAIN, 24);
      g.setFont(font);
      g.drawString("Welcome to TutorialsPoint", 50, 150);
   }
}
```

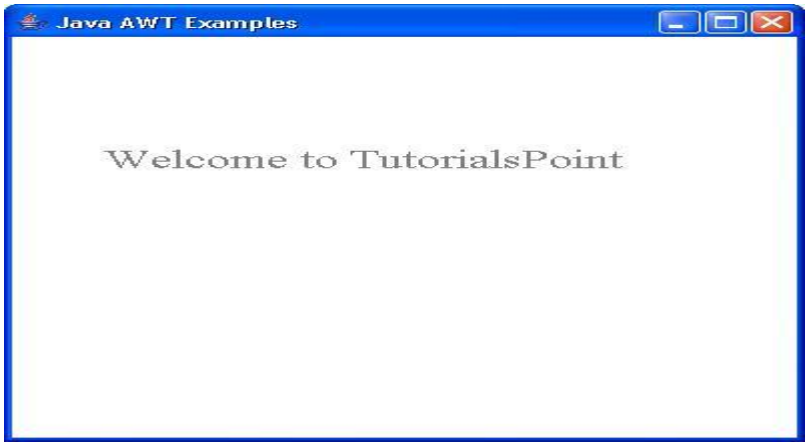
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT Graphics2D Class

The Graphics2D class extends the Graphics class to provide more sophisticated control over geometry, coordinate transformations, color management, and text layout.

Class Declaration

Following is the declaration for **java.awt.Graphics2D** class:

```
public abstract class Graphics2D
   extends Graphics
```

Class Constructors

S.N.	Constructor & Description
1	**Graphics2D()** Constructs a new Graphics2D object.

Class Methods

S.N.	Method & Description
1	**abstract void addRenderingHints(Map hints)** Sets the values of an arbitrary number of preferences for the rendering algorithms.
2	**abstract void clip(Shape s)** Intersects the current Clip with the interior of the specified Shape and sets the Clip to the resulting intersection.
3	**abstract void draw(Shape s)** Strokes the outline of a Shape using the settings of the current Graphics2D context.
4	**void draw3DRect(int x, int y, int width, int height, boolean raised)** Draws a 3-D highlighted outline of the specified rectangle.
5	**abstract void drawGlyphVector(GlyphVector g, float x, float y)** Renders the text of the specified GlyphVector using the Graphics2D context's rendering attributes.
6	**abstract void drawImage(BufferedImage img, BufferedImageOp op, int x, int y)** Renders a BufferedImage that is filtered with a BufferedImageOp.
7	**abstract boolean drawImage(Image img, AffineTransform xform, ImageObserver obs)** Renders an image, applying a transform from image space into user space before drawing.
8	**abstract void drawRenderableImage(RenderableImage img, AffineTransform xform)** Renders a RenderableImage, applying a transform from image space into user space before drawing.
9	**abstract void drawRenderedImage(RenderedImage img, AffineTransform xform)** Renders a RenderedImage, applying a transform from image space into user space before drawing.
10	**abstract void drawString(AttributedCharacterIterator iterator, float x, float y)**

	Renders the text of the specified iterator applying its attributes in accordance with the specification of the TextAttribute class.
11	**abstract void drawString(AttributedCharacterIterator iterator, int x, int y)** Renders the text of the specified iterator applying its attributes in accordance with the specification of the TextAttribute class.
12	**abstract void drawString(String str, float x, float y)** Renders the text specified by the specified String, using the current text attribute state in the Graphics2D context
13	**abstract void drawString(String str, int x, int y)** Renders the text of the specified String, using the current text attribute state in the Graphics2D context.
14	**abstract void fill(Shape s)** Fills the interior of a Shape using the settings of the Graphics2D context.
15	**void fill3DRect(int x, int y, int width, int height, boolean raised)** Paints a 3-D highlighted rectangle filled with the current color.
16	**abstract Color getBackground()** Returns the background color used for clearing a region.
17	**abstract Composite getComposite()** Returns the current Composite in the Graphics2D context.
18	**abstract GraphicsConfiguration getDeviceConfiguration()** Returns the device configuration associated with this Graphics2D.
19	**abstract FontRenderContext getFontRenderContext()** Get the rendering context of the Font within this Graphics2D context.
20	**abstract Paint getPaint()** Returns the current Paint of the Graphics2D context.
21	**abstract Object getRenderingHint(RenderingHints.Key hintKey)** Returns the value of a single preference for the rendering algorithms.
22	**abstract RenderingHints getRenderingHints()** Gets the preferences for the rendering algorithms.
23	**abstract Stroke getStroke()** Returns the current Stroke in the Graphics2D context.
24	**abstract AffineTransform getTransform()** Returns a copy of the current Transform in the Graphics2D context.
25	**abstract boolean hit(Rectangle rect, Shape s, boolean onStroke)** Checks whether or not the specified Shape intersects the specified Rectangle, which is in device space.
26	**abstract void rotate(double theta)** Concatenates the current Graphics2D Transform with a rotation transform.
27	**abstract void rotate(double theta, double x, double y)** Concatenates the current Graphics2D Transform with a translated rotation transform.
28	**abstract void scale(double sx, double sy)** Concatenates the current Graphics2D Transform with a scaling transformation Subsequent rendering is resized according to the specified scaling factors relative to the previous scaling.
29	**abstract void setBackground(Color color)**

	Sets the background color for the Graphics2D context.
30	**abstract void setComposite(Composite comp)** Sets the Composite for the Graphics2D context.
31	**abstract void setPaint(Paint paint)** Sets the Paint attribute for the Graphics2D context.
32	**abstract void setRenderingHint(RenderingHints.Key hintKey, Object hintValue)** Sets the value of a single preference for the rendering algorithms.
33	**abstract void setRenderingHints(Map hints)** Replaces the values of all preferences for the rendering algorithms with the specified hints.
34	**abstract void setStroke(Stroke s)** Sets the Stroke for the Graphics2D context.
35	**abstract void setTransform(AffineTransform Tx)** Overwrites the Transform in the Graphics2D context.
36	**abstract void shear(double shx, double shy)** Concatenates the current Graphics2D Transform with a shearing transform.
37	**abstract void transform(AffineTransform Tx)** Composes an AffineTransform object with the Transform in this Graphics2D according to the rulelast-specified-first-applied.
38	**abstract void translate(double tx, double ty)** Concatenates the current Graphics2D Transform with a translation transform.
39	**abstract void translate(int x, int y)** Translates the origin of the Graphics2D context to the point (x, y) in the current coordinate system.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

Graphics2D Example

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

AWTGraphicsDemo.java

```
package com.tutorialspoint.gui;


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

```
public class AWTGraphicsDemo extends Frame {

   public AWTGraphicsDemo(){
      super("Java AWT Examples");
      prepareGUI();
   }

   public static void main(String[] args){
      AWTGraphicsDemo  awtGraphicsDemo = new AWTGraphicsDemo();
      awtGraphicsDemo.setVisible(true);
   }

   private void prepareGUI(){
      setSize(400,400);
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
   }

   @Override
   public void paint(Graphics g) {
      Graphics2D g2 = (Graphics2D)g;
      g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
         RenderingHints.VALUE_ANTIALIAS_ON);
      Font font = new Font("Serif", Font.PLAIN, 24);
      g2.setFont(font);
      g2.drawString("Welcome to TutorialsPoint", 50, 70);
   }
}
```

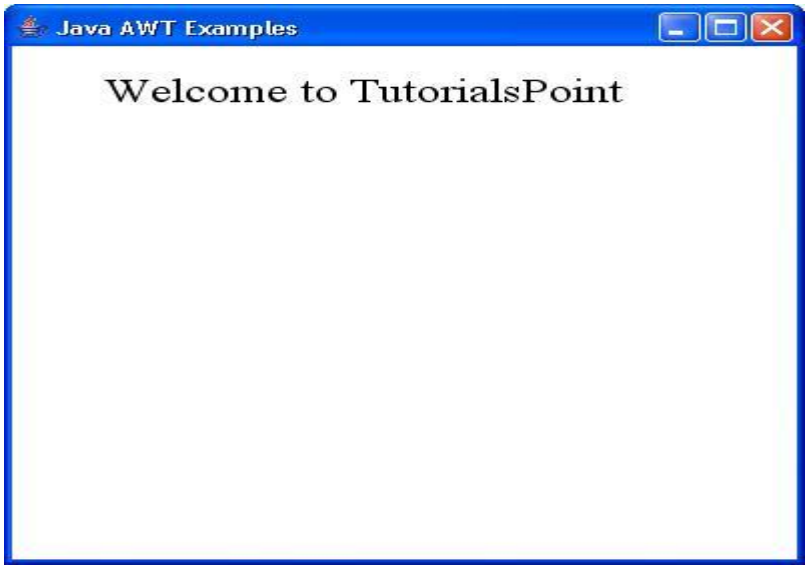
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT Arc2D Class

The Arc2D class is the superclass for all objects that stores a 2D arc defined by a framing rectangle, start angle, angular extent (length of the arc), and a closure type (OPEN, CHORD, or PIE).

Class Declaration

Following is the declaration for **java.awt.Arc2D** class:

```
public abstract class Arc2D
   extends RectangularShape
```

Field

Following are the fields for **java.awt.geom.Arc2D** class:

- **static int CHORD** -- The closure type for an arc closed by drawing a straight line segment from the start of the arc segment to the end of the arc segment.
- **static int OPEN** -- The closure type for an open arc with no path segments connecting the two ends of the arc segment.
- **static int PIE** -- The closure type for an arc closed by drawing straight line segments from the start of the arc segment to the center of the full ellipse and from that point to the end of the arc segment.

Class Constructors

S.N.	Constructor & Description
1	**protected Arc2D(int type)** This is an abstract class that cannot be instantiated directly.

Class Methods

S.N.	Method & Description
1	**boolean contains(double x, double y)** Determines whether or not the specified point is inside the boundary of the arc.
2	**boolean contains(double x, double y, double w, double h)** Determines whether or not the interior of the arc entirely contains the specified rectangle.
3	**boolean contains(Rectangle2D r)** Determines whether or not the interior of the arc entirely contains the specified rectangle.
4	**boolean containsAngle(double angle)** Determines whether or not the specified angle is within the angular extents of the arc.
5	**boolean equals(Object obj)** Determines whether or not the specified Object is equal to this Arc2D.
6	**abstract double getAngleExtent()** Returns the angular extent of the arc.
7	**abstract double getAngleStart()** Returns the starting angle of the arc.
8	**int getArcType()** Returns the arc closure type of the arc: OPEN, CHORD, or PIE.
9	**Rectangle2D getBounds2D()** Returns the high-precision framing rectangle of the arc.
10	**Point2D getEndPoint()** Returns the ending point of the arc.
11	**PathIterator getPathIterator(AffineTransform at)** Returns an iteration object that defines the boundary of the arc.
12	**Point2D getStartPoint()** Returns the starting point of the arc.
13	**int hashCode()**

	Returns the hashcode for this Arc2D.
14	**boolean intersects(double x, double y, double w, double h)** Determines whether or not the interior of the arc intersects the interior of the specified rectangle.
15	**protected abstract Rectangle2D makeBounds(double x, double y, double w, double h)** Constructs a Rectangle2D of the appropriate precision to hold the parameters calculated to be the framing rectangle of this arc.
16	**abstract void setAngleExtent(double angExt)** Sets the angular extent of this arc to the specified double value.
17	**void setAngles(double x1, double y1, double x2, double y2)** Sets the starting angle and angular extent of this arc using two sets of coordinates.
18	**void setAngles(Point2D p1, Point2D p2)** Sets the starting angle and angular extent of this arc using two points.
19	**abstract void setAngleStart(double angSt)** Sets the starting angle of this arc to the specified double value.
20	**void setAngleStart(Point2D p)** Sets the starting angle of this arc to the angle that the specified point defines relative to the center of this arc.
21	**void setArc(Arc2D a)** Sets this arc to be the same as the specified arc.
22	**abstract void setArc(double x, double y, double w, double h, double angSt, double angExt, int closure)** Sets the location, size, angular extents, and closure type of this arc to the specified double values.
23	**void setArc(Point2D loc, Dimension2D size, double angSt, double angExt, int closure)** Sets the location, size, angular extents, and closure type of this arc to the specified values.
24	**void setArc(Rectangle2D rect, double angSt, double angExt, int closure)** Sets the location, size, angular extents, and closure type of this arc to the specified values.
25	**void setArcByCenter(double x, double y, double radius, double angSt, double angExt, int closure)** Sets the position, bounds, angular extents, and closure type of this arc to the specified values.
26	**void setArcByTangent(Point2D p1, Point2D p2, Point2D p3, double radius)** Sets the position, bounds, and angular extents of this arc to the specified value.
27	**void setArcType(int type)** Sets the closure type of this arc to the specified value: OPEN, CHORD, or PIE.
28	**void setFrame(double x, double y, double w, double h)** Sets the location and size of the framing rectangle of this Shape to the specified rectangular values.

Methods Inherited

This class inherits methods from the following classes:

- java.awt.geom.RectangularShape
- java.lang.Object

Arc2D Example

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

AWTGraphicsDemo.java

```
package com.tutorialspoint.gui;

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class AWTGraphicsDemo extends Frame {

   public AWTGraphicsDemo(){
      super("Java AWT Examples");
      prepareGUI();
   }

   public static void main(String[] args){
      AWTGraphicsDemo  awtGraphicsDemo = new AWTGraphicsDemo();
      awtGraphicsDemo.setVisible(true);
   }

   private void prepareGUI(){
      setSize(400,400);
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
```

```
      });
   }

   @Override
   public void paint(Graphics g) {
      Arc2D.Float arc = new Arc2D.Float(Arc2D.PIE);
      arc.setFrame(70, 200, 150, 150);
      arc.setAngleStart(0);
      arc.setAngleExtent(145);
      Graphics2D g2 = (Graphics2D) g;
      g2.setColor(Color.gray);
      g2.draw(arc);
      g2.setColor(Color.red);
      g2.fill(arc);
      g2.setColor(Color.black);
      Font font = new Font("Serif", Font.PLAIN, 24);
      g2.setFont(font);
      g.drawString("Welcome to TutorialsPoint", 50, 70);
      g2.drawString("Arc2D.PIE", 100, 120);
   }
}
```

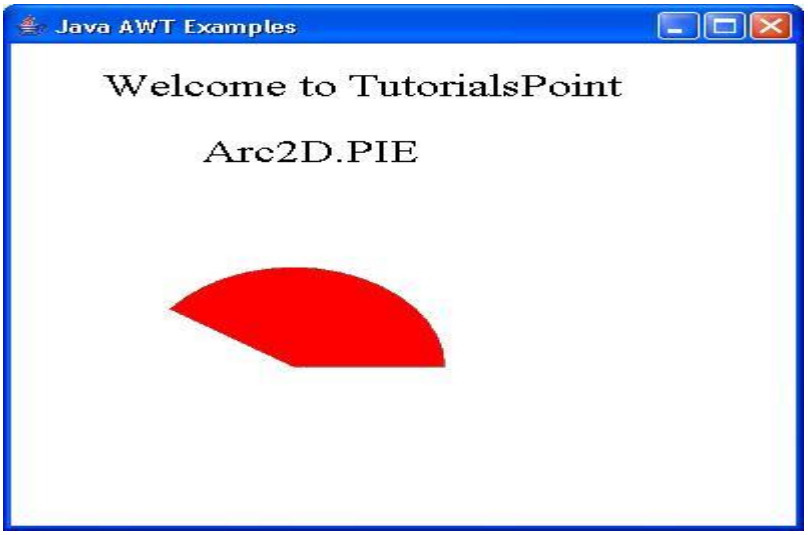
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT CubicCurve2D Class

The CubicCurve2D class states a cubic parametric curve segment in (x & y) coordinate space.

Class Declaration

Following is the declaration for **java.awt.geom.CubicCurve2D** class:

```
public abstract class CubicCurve2D
   extends Object
      implements Shape, Cloneable
```

Class Constructors

S.N.	Constructor & Description
1	**protected CubicCurve2D()** This is an abstract class that cannot be instantiated directly.

Class Methods

S.N.	Method & Description
1	**Object clone()** Creates a new object of the same class as this object.
2	**boolean contains(double x, double y)** Tests if the specified coordinates are inside the boundary of the Shape.
3	**boolean contains(double x, double y, double w, double h)** Tests if the interior of the Shape entirely contains the specified rectangular area.
4	**boolean contains(Point2D p)** Tests if a specified Point2D is inside the boundary of the Shape.

5	**boolean contains(Rectangle2D r)** Tests if the interior of the Shape entirely contains the specified Rectangle2D.
6	**Rectangle getBounds()** Returns an integer Rectangle that completely encloses the Shape.
7	**abstract Point2D getCtrlP1()** Returns the first control point.
8	**abstract Point2D getCtrlP2()** Returns the second control point.
9	**abstract double getCtrlX1()** Returns the X coordinate of the first control point in double precision.
10	**abstract double getCtrlX2()** Returns the X coordinate of the second control point in double precision.
11	**abstract double getCtrlY1()** Returns the Y coordinate of the first control point in double precision.
12	**abstract double getCtrlY2()** Returns the Y coordinate of the second control point in double precision.
13	**double getFlatness()** Returns the flatness of this curve.
14	**static double getFlatness(double[] coords, int offset)** Returns the flatness of the cubic curve specified by the control points stored in the indicated array at the indicated index.
15	**static double getFlatness(double x1, double y1, double ctrlx1, double ctrly1, double ctrlx2, double ctrly2, double x2, double y2)** Returns the flatness of the cubic curve specified by the indicated control points.
16	**double getFlatnessSq()** Returns the square of the flatness of this curve.
17	**static double getFlatnessSq(double[] coords, int offset)** Returns the square of the flatness of the cubic curve specified by the control points stored in the indicated array at the indicated index.
18	**static double getFlatnessSq(double x1, double y1, double ctrlx1, double ctrly1, double ctrlx2, double ctrly2, double x2, double y2)** Returns the square of the flatness of the cubic curve specified by the indicated control points.
19	**abstract Point2D getP1()** Returns the start point.
20	**abstract Point2D getP2()** Returns the end point.
21	**PathIterator getPathIterator(AffineTransform at)** Returns an iteration object that defines the boundary of the shape.
22	**PathIterator getPathIterator(AffineTransform at, double flatness)** Return an iteration object that defines the boundary of the flattened shape.
23	**abstract double getX1()** Returns the X coordinate of the start point in double precision.
24	**abstract double getX2()** Returns the X coordinate of the end point in double precision.

25	**abstract double getY1()** Returns the Y coordinate of the start point in double precision.
26	**abstract double getY2()** Returns the Y coordinate of the end point in double precision.
27	**boolean intersects(double x, double y, double w, double h)** Tests if the interior of the Shape intersects the interior of a specified rectangular area.
28	**boolean intersects(Rectangle2D r)** Tests if the interior of the Shape intersects the interior of a specified Rectangle2D.
29	**void setCurve(CubicCurve2D c)** Sets the location of the end points and control points of this curve to the same as those in the specified CubicCurve2D.
30	**void setCurve(double[] coords, int offset)** Sets the location of the end points and control points of this curve to the double coordinates at the specified offset in the specified array.
31	**abstract void setCurve(double x1, double y1, double ctrlx1, double ctrly1, double ctrlx2, double ctrly2, double x2, double y2)** Sets the location of the end points and control points of this curve to the specified double coordinates.
32	**void setCurve(Point2D[] pts, int offset)** Sets the location of the end points and control points of this curve to the coordinates of the Point2D objects at the specified offset in the specified array.
33	**void setCurve(Point2D p1, Point2D cp1, Point2D cp2, Point2D p2)** Sets the location of the end points and control points of this curve to the specified Point2D coordinates.
34	**static int solveCubic(double[] eqn)** Solves the cubic whose coefficients are in the eqn array and places the non-complex roots back into the same array, returning the number of roots.
35	**static int solveCubic(double[] eqn, double[] res)** Solve the cubic whose coefficients are in the eqn array and place the non-complex roots into the res array, returning the number of roots.
36	**void subdivide(CubicCurve2D left, CubicCurve2D right)** Subdivides this cubic curve and stores the resulting two subdivided curves into the left and right curve parameters.
37	**static void subdivide(CubicCurve2D src, CubicCurve2D left, CubicCurve2D right)** Subdivides the cubic curve specified by the src parameter and stores the resulting two subdivided curves into the left and right curve parameters.
38	**static void subdivide(double[] src, int srcoff, double[] left, int leftoff, double[] right, int rightoff)** Subdivides the cubic curve specified by the coordinates stored in the src array at indices srcoff through (srcoff + 7) and stores the resulting two subdivided curves into the two result arrays at the corresponding indices.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

CubicCurve2D Example

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

AWTGraphicsDemo.java

```
package com.tutorialspoint.gui;

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class AWTGraphicsDemo extends Frame {

   public AWTGraphicsDemo(){
      super("Java AWT Examples");
      prepareGUI();
   }

   public static void main(String[] args){
      AWTGraphicsDemo  awtGraphicsDemo = new AWTGraphicsDemo();
      awtGraphicsDemo.setVisible(true);
   }

   private void prepareGUI(){
      setSize(400,400);
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
```

```
      });
   }

   @Override
   public void paint(Graphics g) {
      CubicCurve2D shape = new CubicCurve2D.Float();
      shape.setCurve(250F,250F,20F,90F,140F,100F,350F,330F);
      Graphics2D g2 = (Graphics2D) g;
      g2.draw (shape);
      Font font = new Font("Serif", Font.PLAIN, 24);
      g2.setFont(font);
      g.drawString("Welcome to TutorialsPoint", 50, 70);
      g2.drawString("CubicCurve2D.Curve", 100, 120);
   }
}
```

Compile the program using command prompt. Go to **D:/ > AWT**

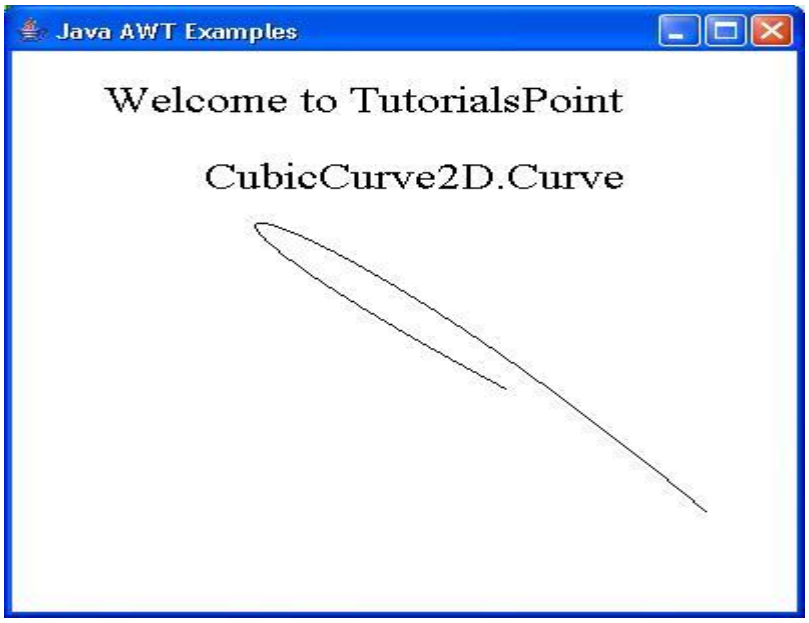
and type the following command:

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

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

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

Output



AWT Ellipse2D Class

The Ellipse2D class states an ellipse that is defined by a framing rectangle.

Class Declaration

Following is the declaration for **java.awt.geom.Ellipse2D** class:

```
public abstract class Ellipse2D
   extends RectangularShape
```

Class Constructors

S.N.	Constructor & Description
1	**protected Ellipse2D()** This is an abstract class that cannot be instantiated directly.

Class Methods

S.N.	Method & Description
1	**boolean contains(double x, double y)** Tests if the specified coordinates are inside the boundary of the Shape.
2	**boolean contains(double x, double y, double w, double h)** Tests if the interior of the Shape entirely contains the specified rectangular area.
3	**boolean equals(Object obj)** Determines whether or not the specified Object is equal to this Ellipse2D.

4	**PathIterator getPathIterator(AffineTransform at)** Returns an iteration object that defines the boundary of this Ellipse2D.
5	**int hashCode()** Returns the hashcode for this Ellipse2D.
6	**boolean intersects(double x, double y, double w, double h)** Tests if the interior of the Shape intersects the interior of a specified rectangular area.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

Ellipse2D Example

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

AWTGraphicsDemo.java

```
package com.tutorialspoint.gui;

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class AWTGraphicsDemo extends Frame {

   public AWTGraphicsDemo(){
      super("Java AWT Examples");
      prepareGUI();
   }

   public static void main(String[] args){
      AWTGraphicsDemo  awtGraphicsDemo = new AWTGraphicsDemo();
      awtGraphicsDemo.setVisible(true);
   }

   private void prepareGUI(){
```

```
      setSize(400,400);
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
   }

   @Override
   public void paint(Graphics g) {
      Ellipse2D shape = new Ellipse2D.Float();
      shape.setFrame(100, 150, 200,100);
      Graphics2D g2 = (Graphics2D) g;
      g2.draw (shape);
      Font font = new Font("Serif", Font.PLAIN, 24);
      g2.setFont(font);
      g.drawString("Welcome to TutorialsPoint", 50, 70);
      g2.drawString("Ellipse2D.Oval", 100, 120);
   }
}
```

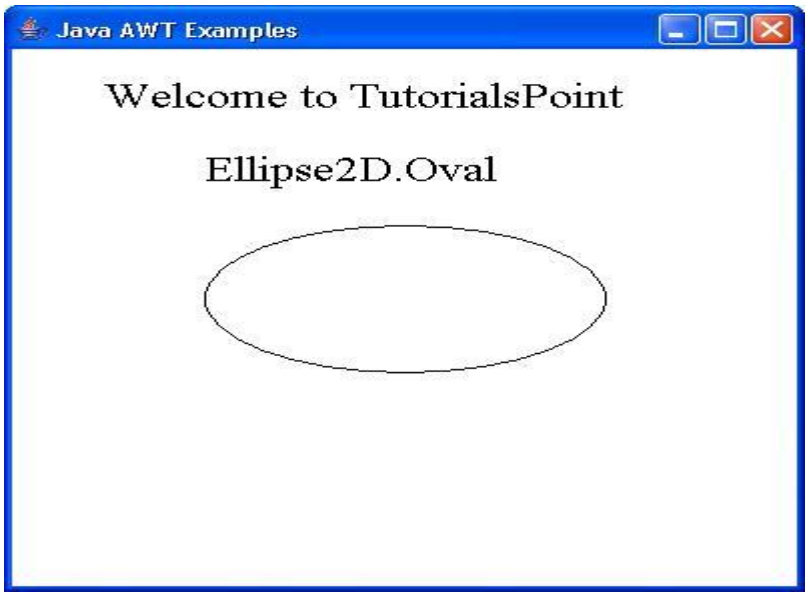
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT Rectangle2D Class

The Rectangle2D class states a rectangle defined by a location (x & y) and dimension (w x h).

Class Declaration

Following is the declaration for **java.awt.geom.Rectangle2D** class:

```
public abstract class Rectangle2D
   extends RectangularShape
```

Field

Following are the fields for **java.awt.geom.Arc2D** class:

- **static int OUT_BOTTOM** -- The bitmask that indicates that a point lies below this Rectangle2D.
- **static int OUT_LEFT** -- The bitmask that indicates that a point lies to the left of this Rectangle2D.
- **static int OUT_RIGHT** -- The bitmask that indicates that a point lies to the right of this Rectangle2D.
- **static int OUT_TOP** -- The bitmask that indicates that a point lies above this Rectangle2D.

Class Constructors

S.N.	Constructor & Description
1	**protected Rectangle2D()** This is an abstract class that cannot be instantiated directly.

Class Methods

S.N.	Method & Description
1	**void add(double newx, double newy)** Adds a point, specified by the double precision arguments newx and newy, to this Rectangle2D.
2	**void add(Point2D pt)** Adds the Point2D object pt to this Rectangle2D.
3	**void add(Rectangle2D r)** Adds a Rectangle2D object to this Rectangle2D.
4	**boolean contains(double x, double y)** Tests if the specified coordinates are inside the boundary of the Shape.
5	**boolean contains(double x, double y, double w, double h)** Tests if the interior of the Shape entirely contains the specified rectangular area.
6	**abstract Rectangle2D createIntersection(Rectangle2D r)** Returns a new Rectangle2D object representing the intersection of this Rectangle2D with the specified Rectangle2D.
7	**abstract Rectangle2D createUnion(Rectangle2D r)** Returns a new Rectangle2D object representing the union of this Rectangle2D with the specified Rectangle2D.
8	**boolean equals(Object obj)** Determines whether or not the specified Object is equal to this Rectangle2D.
9	**Rectangle2D getBounds2D()** Returns a high precision and more accurate bounding box of the Shape than the getBounds method.
10	**PathIterator getPathIterator(AffineTransform at)** Returns an iteration object that defines the boundary of this Rectangle2D.
11	**PathIterator getPathIterator(AffineTransform at, double flatness)** Returns an iteration object that defines the boundary of the flattened Rectangle2D.
12	**int hashCode()** Returns the hashcode for this Rectangle2D.
13	**static void intersect(Rectangle2D src1, Rectangle2D src2, Rectangle2D dest)** Intersects the pair of specified source Rectangle2D objects and puts the result into the specified destination Rectangle2D object.
14	**boolean intersects(double x, double y, double w, double h)** Tests if the interior of the Shape intersects the interior of a specified rectangular area.

15	**boolean intersectsLine(double x1, double y1, double x2, double y2)** Tests if the specified line segment intersects the interior of this Rectangle2D.
16	**boolean intersectsLine(Line2D l)** Tests if the specified line segment intersects the interior of this Rectangle2D.
17	**abstract int outcode(double x, double y)** Determines where the specified coordinates lie with respect to this Rectangle2D.
18	**int outcode(Point2D p)** Determines where the specified Point2D lies with respect to this Rectangle2D.
19	**void setFrame(double x, double y, double w, double h)** Sets the location and size of the outer bounds of this Rectangle2D to the specified rectangular values.
20	**abstract void setRect(double x, double y, double w, double h)** Sets the location and size of this Rectangle2D to the specified double values.
21	**void setRect(Rectangle2D r)** Sets this Rectangle2D to be the same as the specified Rectangle2D.
22	**static void union(Rectangle2D src1, Rectangle2D src2, Rectangle2D dest)** Unions the pair of source Rectangle2D objects and puts the result into the specified destination Rectangle2D object.

Methods inherited

This class inherits methods from the following classes:

- java.awt.geom.RectangularShape
- java.lang.Object

Ellipse2D Example

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

AWTGraphicsDemo.java

```
package com.tutorialspoint.gui;


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

```
public class AWTGraphicsDemo extends Frame {

   public AWTGraphicsDemo(){
      super("Java AWT Examples");
      prepareGUI();
   }

   public static void main(String[] args){
      AWTGraphicsDemo  awtGraphicsDemo = new AWTGraphicsDemo();
      awtGraphicsDemo.setVisible(true);
   }

   private void prepareGUI(){
      setSize(400,400);
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
   }

   @Override
   public void paint(Graphics g) {
      Rectangle2D shape = new Rectangle2D.Float();
      shape.setFrame(100, 150, 200,100);
      Graphics2D g2 = (Graphics2D) g;
      g2.draw (shape);
      Font font = new Font("Serif", Font.PLAIN, 24);
      g2.setFont(font);
      g.drawString("Welcome to TutorialsPoint", 50, 70);
      g2.drawString("Rectangle2D.Rectangle", 100, 120);
   }
```

```
}
```

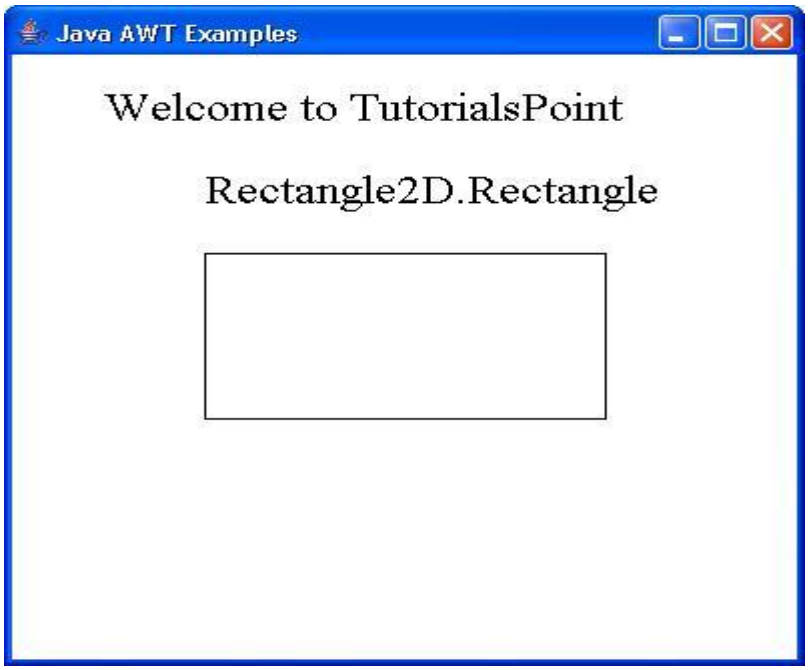
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT QuadCurve2D Class

The QuadCurve2D class states a quadratic parametric curve segment in (x & y) coordinate space.

Class Declaration

Following is the declaration for **java.awt.geom.QuadCurve2D** class:

```
public abstract class QuadCurve2D
   extends Object
      implements Shape, Cloneable
```

Class Constructors

S.N.	Constructor & Description
1	**protected QuadCurve2D() ()** This is an abstract class that cannot be instantiated directly.

Class Methods

S.N.	Method & Description
1	**Object clone()** Creates a new object of the same class and with the same contents as this object.
2	**boolean contains(double x, double y)** Tests if the specified coordinates are inside the boundary of the Shape.
3	**boolean contains(double x, double y, double w, double h)** Tests if the interior of the Shape entirely contains the specified rectangular area.
4	**boolean contains(Point2D p)** Tests if a specified Point2D is inside the boundary of the Shape.
5	**boolean contains(Rectangle2D r)** Tests if the interior of the Shape entirely contains the specified Rectangle2D.
6	**Rectangle getBounds()** Returns an integer Rectangle that completely encloses the Shape.
7	**abstract Point2D getCtrlPt()** Returns the control point.
8	**abstract double getCtrlX()** Returns the X coordinate of the control point in double precision.
9	**abstract double getCtrlY()** Returns the Y coordinate of the control point in double precision.
10	**doublegetFlatness()** Returns the flatness, or maximum distance of a control point from the line connecting the end points, of this QuadCurve2D.
11	**static double getFlatness(double[] coords, int offset)** Returns the flatness, or maximum distance of a control point from the line connecting the end points, of the quadratic curve specified by the control points stored in the indicated array at the indicated index.
12	**static double getFlatness(double x1, double y1, double ctrlx, double ctrly, double x2, double y2)** Returns the flatness, or maximum distance of a control point from the line connecting the end points, of the quadratic curve specified by the indicated control points.
13	**double getFlatnessSq()** Returns the square of the flatness, or maximum distance of a control point from the line connecting the end points, of this QuadCurve2D.
14	**static double getFlatnessSq(double[] coords, int offset)** Returns the square of the flatness, or maximum distance of a control point from the line connecting the end points, of the quadratic curve

	specified by the control points stored in the indicated array at the indicated index.
15	**static double getFlatnessSq(double x1, double y1, double ctrlx, double ctrly, double x2, double y2)** Returns the square of the flatness, or maximum distance of a control point from the line connecting the end points, of the quadratic curve specified by the indicated control points.
16	**abstract Point2D getP1()** Returns the start point.
17	**abstract Point2D getP2()** Returns the end point.
18	**PathIterator getPathIterator(AffineTransform at)** Returns an iteration object that defines the boundary of the shape of this QuadCurve2D.
19	**PathIterator getPathIterator(AffineTransform at, double flatness)** Returns an iteration object that defines the boundary of the flattened shape of this QuadCurve2D.
20	**abstract double getX1()** Returns the X coordinate of the start point in double in precision.
21	**abstract double getX2()** Returns the X coordinate of the end point in double precision.
22	**abstract double getY1()** Returns the Y coordinate of the start point in double precision.
23	**abstract double getY2()** Returns the Y coordinate of the end point in double precision.
24	**boolean intersects(double x, double y, double w, double h)** Tests if the interior of the Shape intersects the interior of a specified rectangular area.
25	**boolean intersects(Rectangle2D r)** Tests if the interior of the Shape intersects the interior of a specified Rectangle2D.
26	**void setCurve(double[] coords, int offset)** Sets the location of the end points and control points of this QuadCurve2D to the double coordinates at the specified offset in the specified array.
27	**abstract void setCurve(double x1, double y1, double ctrlx, double ctrly, double x2, double y2)** Sets the location of the end points and control point of this curve to the specified double coordinates.
28	**void setCurve(Point2D[] pts, int offset)** Sets the location of the end points and control points of this QuadCurve2D to the coordinates of the Point2D objects at the specified offset in the specified array.
29	**void setCurve(Point2D p1, Point2D cp, Point2D p2)** Sets the location of the end points and control point of this QuadCurve2D to the specified Point2D coordinates.
30	**void setCurve(QuadCurve2D c)**

	Sets the location of the end points and control point of this QuadCurve2D to the same as those in the specified QuadCurve2D.
31	**static int solveQuadratic(double[] eqn)** Solves the quadratic whose coefficients are in the eqn array and places the non-complex roots back into the same array, returning the number of roots.
32	**static int solveQuadratic(double[] eqn, double[] res)** Solves the quadratic whose coefficients are in the eqn array and places the non-complex roots into the res array, returning the number of roots.
33	**static void subdivide(double[] src, int srcoff, double[] left, int leftoff, double[] right, int rightoff)** Subdivides the quadratic curve specified by the coordinates stored in the src array at indices srcoff through srcoff + 5 and stores the resulting two subdivided curves into the two result arrays at the corresponding indices.
34	**void subdivide(QuadCurve2D left, QuadCurve2D right)** Subdivides this QuadCurve2D and stores the resulting two subdivided curves into the left and right curve parameters.
35	**static void subdivide(QuadCurve2D src, QuadCurve2D left, QuadCurve2D right)** Subdivides the quadratic curve specified by the src parameter and stores the resulting two subdivided curves into the left and right curve parameters.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

QuadCurve2D Example

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

AWTGraphicsDemo

```
package com.tutorialspoint.gui;


import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;


public class AWTGraphicsDemo extends Frame {
```

```
   public AWTGraphicsDemo(){

      super("Java AWT Examples");

      prepareGUI();
   }

   public static void main(String[] args){
      AWTGraphicsDemo  awtGraphicsDemo = new AWTGraphicsDemo();
      awtGraphicsDemo.setVisible(true);
   }

   private void prepareGUI(){
      setSize(400,400);
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
   }

   @Override
   public void paint(Graphics g) {
      QuadCurve2D shape = new QuadCurve2D.Double();
      shape.setCurve(250D,250D,100D,100D,200D,150D);
      Graphics2D g2 = (Graphics2D) g;
      g2.draw (shape);
      Font font = new Font("Serif", Font.PLAIN, 24);
      g2.setFont(font);
      g.drawString("Welcome to TutorialsPoint", 50, 70);
      g2.drawString("QuadCurve2D.Curve", 100, 120);
   }
}
```

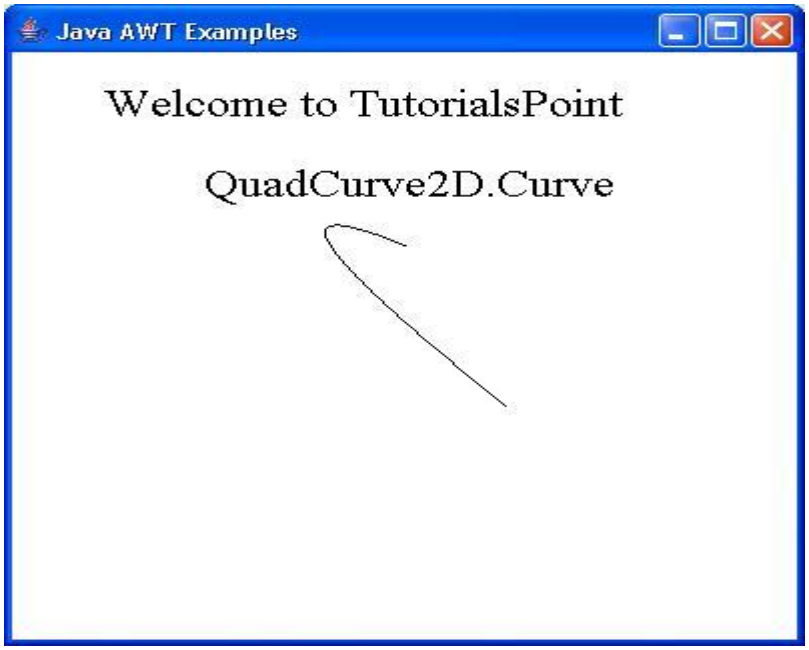
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT Line2D Class

The Line2D class states a line segment in (x & y) coordinate space.

Class Declaration

Following is the declaration for **java.awt.geom.Line2D** class:

```
public abstract class Line2D
   extends Object
      implements Shape, Cloneable
```

Class Constructors

S.N.	Constructor & Description
1	**protected Line2D() ()** This is an abstract class that cannot be instantiated directly.

Class Methods

S.N.	Method & Description
1	**Object clone()** Creates a new object of the same class as this object.
2	**boolean contains(double x, double y)** Tests if a specified coordinate is inside the boundary of this Line2D.
3	**boolean contains(double x, double y, double w, double h)** Tests if the interior of this Line2D entirely contains the specified set of rectangular coordinates.
4	**boolean contains(Point2D p)** Tests if a given Point2D is inside the boundary of this Line2D.
5	**boolean contains(Rectangle2D r)** Tests if the interior of this Line2D entirely contains the specified Rectangle2D.
6	**Rectangle getBounds()** Returns an integer Rectangle that completely encloses the Shape.
7	**abstract Point2D getP1()** Returns the start Point2D of this Line2D.
8	**abstract Point2D getP2()** Returns the end Point2D of this Line2D.
9	**PathIterator getPathIterator(AffineTransform at)** Returns an iteration object that defines the boundary of this Line2D.
10	**PathIterator getPathIterator(AffineTransform at, double flatness)** Returns an iteration object that defines the boundary of this flattened Line2D.
11	**abstract double getX1()** Returns the X coordinate of the start point in double precision.
12	**abstract double getX2()** Returns the X coordinate of the end point in double precision.
13	**abstract double getY1()** Returns the Y coordinate of the start point in double precision.
14	**abstract double getY2()** Returns the Y coordinate of the end point in double precision.
15	**boolean intersects(double x, double y, double w, double h)** Tests if the interior of the Shape intersects the interior of a specified rectangular area.
16	**boolean intersects(Rectangle2D r)** Tests if the interior of the Shape intersects the interior of a specified Rectangle2D.
17	**boolean intersectsLine(double x1, double y1, double x2, double y2)** Tests if the line segment from (x1,y1) to (x2,y2) intersects this line segment.
18	**boolean intersectsLine(Line2D l)** Tests if the specified line segment intersects this line segment.
19	**static boolean linesIntersect(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4)**

	Tests if the line segment from (x1,y1) to (x2,y2) intersects the line segment from (x3,y3) to (x4,y4).
20	**double ptLineDist(double px, double py)** Returns the distance from a point to this line.
21	**static double ptLineDist(double x1, double y1, double x2, double y2, double px, double py)** Returns the distance from a point to a line.
22	**double ptLineDist(Point2D pt)** Returns the distance from a Point2D to this line.
23	**double ptLineDistSq(double px, double py)** Returns the square of the distance from a point to this line.
24	**static double ptLineDistSq(double x1, double y1, double x2, double y2, double px, double py)** Returns the square of the distance from a point to a line.
25	**double ptLineDistSq(Point2D pt)** Returns the square of the distance from a specified Point2D to this line.
26	**double ptSegDist(double px, double py)** Returns the distance from a point to this line segment.
27	**static double ptSegDist(double x1, double y1, double x2, double y2, double px, double py)** Returns the distance from a point to a line segment.
28	**double ptSegDist(Point2D pt)** Returns the distance from a Point2D to this line segment.
29	**double ptSegDistSq(double px, double py)** Returns the square of the distance from a point to this line segment.
30	**static double ptSegDistSq(double x1, double y1, double x2, double y2, double px, double py)** Returns the square of the distance from a point to a line segment.
31	**double ptSegDistSq(Point2D pt)** Returns the square of the distance from a Point2D to this line segment.
32	**int relativeCCW(double px, double py)** Returns an indicator of where the specified point (px,py) lies with respect to this line segment.
33	**static int relativeCCW(double x1, double y1, double x2, double y2, double px, double py)** Returns an indicator of where the specified point (px,py) lies with respect to the line segment from (x1,y1) to (x2,y2).
34	**int relativeCCW(Point2D p)** Returns an indicator of where the specified Point2D lies with respect to this line segment.
35	**abstract void setLine(double x1, double y1, double x2, double y2)** Sets the location of the end points of this Line2D to the specified double coordinates.
36	**void setLine(Line2D l)** Sets the location of the end points of this Line2D to the same as those end points of the specified Line2D.
37	**void setLine(Point2D p1, Point2D p2)**

	Sets the location of the end points of this Line2D to the specified Point2D coordinates.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

Line2D Example

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

AWTGraphicsDemo.java

```
package com.tutorialspoint.gui;

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class AWTGraphicsDemo extends Frame {

   public AWTGraphicsDemo(){
      super("Java AWT Examples");
      prepareGUI();
   }

   public static void main(String[] args){
      AWTGraphicsDemo  awtGraphicsDemo = new AWTGraphicsDemo();
      awtGraphicsDemo.setVisible(true);
   }

   private void prepareGUI(){
      setSize(400,400);
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
```

```
            System.exit(0);
         }
      });
   }

   @Override
   public void paint(Graphics g) {
      Line2D shape = new Line2D.Double();
      shape.setLine(250D,250D,150D,150D);
      Graphics2D g2 = (Graphics2D) g;
      g2.draw (shape);
      Font font = new Font("Serif", Font.PLAIN, 24);
      g2.setFont(font);
      g.drawString("Welcome to TutorialsPoint", 50, 70);
      g2.drawString("Line2D.Line", 100, 120);
   }
}
```

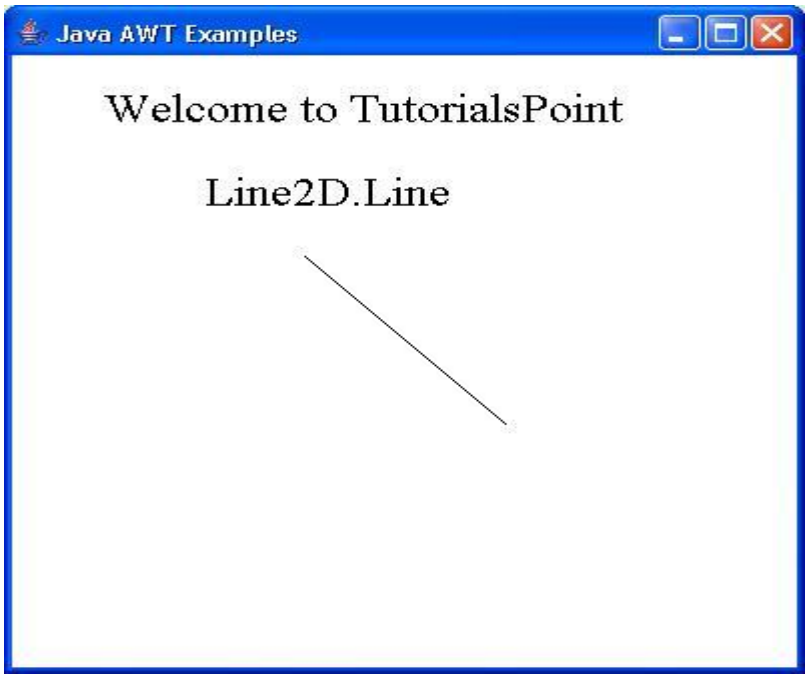
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT Font Class

The Font class states fonts, which are used to render text in a visible way.

Class Declaration

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

```
public class Font
   extends Object
      implements Serializable
```

Field

Following are the fields for **java.awt.geom.Arc2D** class:

- **static int BOLD** -- The bold style constant.
- **static int CENTER_BASELINE** --The baseline used in ideographic scripts like Chinese, Japanese, and Korean when laying out text.
- **static String DIALOG** --A String constant for the canonical family name of the logical font "Dialog".
- **static String DIALOG_INPUT** --A String constant for the canonical family name of the logical font "DialogInput".

- **static int HANGING_BASELINE** -- The baseline used in Devanigiri and similar scripts when laying out text.
- **static int ITALIC** -- The italicized style constant.
- **static int LAYOUT_LEFT_TO_RIGHT** -- A flag to layoutGlyphVector indicating that text is left-to-right as determined by Bidi analysis.
- **static int LAYOUT_NO_LIMIT_CONTEXT** -- A flag to layoutGlyphVector indicating that text in the char array after the indicated limit should not be examined.
- **static int LAYOUT_NO_START_CONTEXT** -- A flag to layoutGlyphVector indicating that text in the char array before the indicated start should not be examined.
- **static int LAYOUT_RIGHT_TO_LEFT** -- A flag to layoutGlyphVector indicating that text is right-to-left as determined by Bidi analysis.
- **static String MONOSPACED** -- A String constant for the canonical family name of the logical font "Monospaced".
- **protected String name** -- The logical name of this Font, as passed to the constructor.
- **static int PLAIN** --The plain style constant.
- **protected float pointSize** -- The point size of this Font in float.
- **static int ROMAN_BASELINE** --The baseline used in most Roman scripts when laying out text.
- **static String SANS_SERIF** -- A String constant for the canonical family name of the logical font "SansSerif".
- **static String SERIF** -- A String constant for the canonical family name of the logical font "Serif".
- **protected int size** --The point size of this Font, rounded to integer.
- **protected int style** -- The style of this Font, as passed to the constructor.
- **static int TRUETYPE_FONT** -- Identify a font resource of type TRUETYPE.
- **static int TYPE1_FONT** -- Identify a font resource of type TYPE1.

Class Constructors

S.N.	Constructor & Description
1	**protected Font() ()** Creates a new Font from the specified font.
2	**Font(Map<? extends AttributedCharacterIterator.Attribute,?> attributes)** Creates a new Font from the specified font.

3	**Font(String name, int style, int size)** Creates a new Font from the specified font.

Class Methods

S.N.	Method & Description
1	**boolean canDisplay(char c)** Checks if this Font has a glyph for the specified character.
2	**boolean canDisplay(int codePoint)** Checks if this Font has a glyph for the specified character.
3	**int canDisplayUpTo(char[] text, int start, int limit)** Indicates whether or not this Font can display the characters in the specified text starting at start and ending at limit.
4	**int canDisplayUpTo(CharacterIterator iter, int start, int limit)** Indicates whether or not this Font can display the text specified by the iter starting at start and ending at limit.
5	**int canDisplayUpTo(String str)** Indicates whether or not this Font can display a specified String.
6	**static Font createFont(int fontFormat, File fontFile)** Returns a new Font using the specified font type and the specified font file.
7	**static Font createFont(int fontFormat, InputStream fontStream)** Returns a new Font using the specified font type and input data.
8	**GlyphVector createGlyphVector(FontRenderContext frc, char[] chars)** Creates a GlyphVector by mapping characters to glyphs one-to-one based on the Unicode cmap in this Font.
9	**GlyphVector createGlyphVector(FontRenderContext frc, CharacterIterator ci)** Creates a GlyphVector by mapping the specified characters to glyphs one-to-one based on the Unicode cmap in this Font.
10	**GlyphVector createGlyphVector(FontRenderContext frc, int[] glyphCodes)** Creates a GlyphVector by mapping characters to glyphs one-to-one based on the Unicode cmap in this Font.
11	**GlyphVector createGlyphVector(FontRenderContext frc, String str)** Creates a GlyphVector by mapping characters to glyphs one-to-one based on the Unicode cmap in this Font.
12	**static Font decode(String str)** Returns the Font that the str argument describes.
13	**Font deriveFont(AffineTransform trans)** Creates a new Font object by replicating the current Font object and applying a new transform to it.
14	**Font deriveFont(float size)** Creates a new Font object by replicating the current Font object and applying a new size to it.
15	**Font deriveFont(int style)**

	Creates a new Font object by replicating the current Font object and applying a new style to it.
16	**Font deriveFont(int style, AffineTransform trans)** Creates a new Font object by replicating this Font object and applying a new style and transform.
17	**Font deriveFont(int style, float size)** Creates a new Font object by replicating this Font object and applying a new style and size.
18	**Font deriveFont(Map<? extends AttributedCharacterIterator.Attribute,?> attributes)** Creates a new Font object by replicating the current Font object and applying a new set of font attributes to it.
19	**boolean equals(Object obj)** Compares this Font object to the specified Object.
20	**protected void finalize()** Disposes the native Font object.
21	**Map<TextAttribute,?> getAttributes()** Returns a map of font attributes available in this Font.
22	**AttributedCharacterIterator.Attribute[] getAvailableAttributes()** Returns the keys of all the attributes supported by this Font.
23	**byte getBaselineFor(char c)** Returns the baseline appropriate for displaying this character.
24	**String getFamily()** Returns the family name of this Font.
25	**String getFamily(Locale l)** Returns the family name of this Font, localized for the specified locale.
26	**static Font getFont(Map<? extends AttributedCharacterIterator.Attribute,?> attributes)** Returns a Font appropriate to the attributes.
27	**static Font getFont(String nm)** Returns a Font object fom the system properties list.
28	**static Font getFont(String nm, Font font)** Gets the specified Font from the system properties list.
29	**String getFontName()** Returns the font face name of this Font.
30	**String getFontName(Locale l)** Returns the font face name of the Font, localized for the specified locale.
31	**float getItalicAngle()** Returns the italic angle of this Font.
32	**LineMetrics getLineMetrics(char[] chars, int beginIndex, int limit, FontRenderContext frc)** Returns a LineMetrics object created with the specified arguments.
33	**LineMetrics getLineMetrics(CharacterIterator ci, int beginIndex, int limit, FontRenderContext frc)** Returns a LineMetrics object created with the specified arguments.
34	**LineMetrics getLineMetrics(String str, FontRenderContext frc)** Returns a LineMetrics object created with the specified String and FontRenderContext.

35	**LineMetrics getLineMetrics(String str, int beginIndex, int limit, FontRenderContext frc)** Returns a LineMetrics object created with the specified arguments.
36	**Rectangle2D getMaxCharBounds(FontRenderContext frc)** Returns the bounds for the character with the maximum bounds as defined in the specified FontRenderContext.
37	**int getMissingGlyphCode()** Returns the glyphCode which is used when this Font does not have a glyph for a specified unicode code point.
38	**String getName()** Returns the logical name of this Font.
39	**int getNumGlyphs()** Returns the number of glyphs in this Font.
40	**java.awt.peer.FontPeer getPeer()** Deprecated. Font rendering is now platform independent.
41	**String getPSName()** Returns the postscript name of this Font.
42	**int getSize()** Returns the point size of this Font, rounded to an integer.
43	**float getSize2D()** Returns the point size of this Font in float value.
44	**Rectangle2D getStringBounds(char[] chars, int beginIndex, int limit, FontRenderContext frc)** Returns the logical bounds of the specified array of characters in the specified FontRenderContext.
45	**Rectangle2D getStringBounds(CharacterIterator ci, int beginIndex, int limit, FontRenderContext frc)** Returns the logical bounds of the characters indexed in the specified CharacterIterator in the specified FontRenderContext.
46	**Rectangle2D getStringBounds(String str, FontRenderContext frc)** Returns the logical bounds of the specified String in the specified FontRenderContext.
47	**Rectangle2D getStringBounds(String str, int beginIndex, int limit, FontRenderContext frc)** Returns the logical bounds of the specified String in the specified FontRenderContext.
48	**int getStyle()** Returns the style of this Font.
49	**AffineTransform getTransform()** Returns a copy of the transform associated with this Font.
50	**int hashCode()** Returns a hashcode for this Font.
51	**boolean hasLayoutAttributes()** Return true if this Font contains attributes that require extra layout processing.
52	**boolean hasUniformLineMetrics()** Checks whether or not this Font has uniform line metrics.
53	**boolean isBold()** Indicates whether or not this Font object's style is BOLD.

54	**boolean isItalic()** Indicates whether or not this Font object's style is ITALIC.
55	**boolean isPlain()** Indicates whether or not this Font object's style is PLAIN.
56	**boolean isTransformed()** Indicates whether or not this Font object has a transform that affects its size in addition to the Size attribute.
57	**GlyphVector layoutGlyphVector(FontRenderContext frc, char[] text, int start, int limit, int flags)** Returns a new GlyphVector object, performing full layout of the text if possible.
58	**String toString()** Converts this Font object to a String representation.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

Font Example

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

AWTGraphicsDemo.java

```
package com.tutorialspoint.gui;

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class AWTGraphicsDemo extends Frame {

   public AWTGraphicsDemo(){
      super("Java AWT Examples");
      prepareGUI();
   }

   public static void main(String[] args){
      AWTGraphicsDemo  awtGraphicsDemo = new AWTGraphicsDemo();
```

```
      awtGraphicsDemo.setVisible(true);
   }

   private void prepareGUI(){
      setSize(400,400);
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
   }

   @Override
   public void paint(Graphics g) {
      Graphics2D g2 = (Graphics2D)g;
      Font plainFont = new Font("Serif", Font.PLAIN, 24);
      g2.setFont(plainFont);
      g2.drawString("Welcome to TutorialsPoint", 50, 70);
      Font italicFont = new Font("Serif", Font.ITALIC, 24);
      g2.setFont(italicFont);
      g2.drawString("Welcome to TutorialsPoint", 50, 120);
      Font boldFont = new Font("Serif", Font.BOLD, 24);
      g2.setFont(boldFont);
      g2.drawString("Welcome to TutorialsPoint", 50, 170);
      Font boldItalicFont = new Font("Serif", Font.BOLD+Font.ITALIC,
24);
      g2.setFont(boldItalicFont);
      g2.drawString("Welcome to TutorialsPoint", 50, 220);
   }
}
```

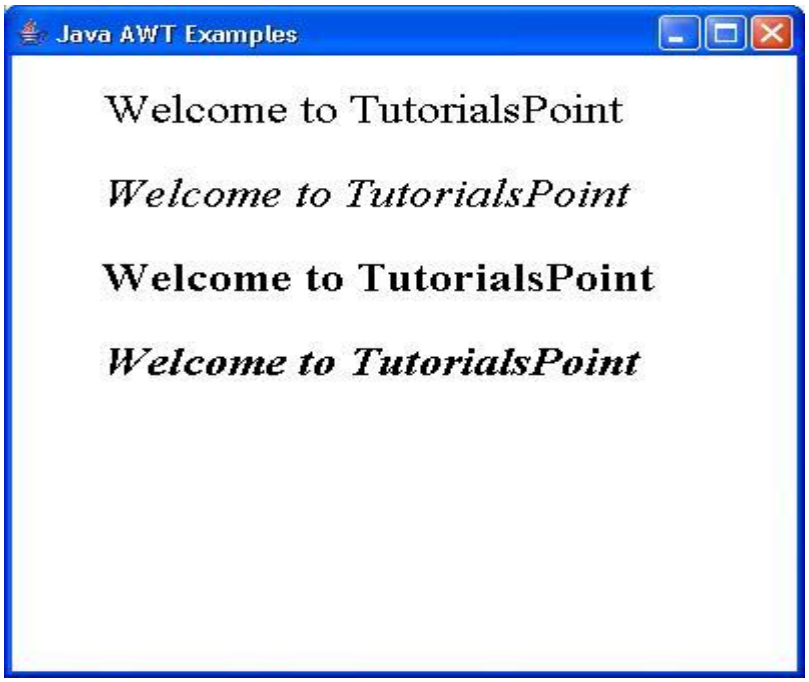
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT Color Class

The Color class states colors in the default sRGB color space or colors in the arbitrary color spaces identified by a ColorSpace.

Class Declaration

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

```
public class Color
   extends Object
      implements Paint, Serializable
```

Field

Following are the fields for **java.awt.geom.Arc2D** class:

- **static Color black** -- The color black.
- **static Color BLACK** -- The color black.

- **static Color blue** -- The color blue.
- **static Color BLUE** -- The color blue.
- **static Color cyan** -- The color cyan.
- **static Color CYAN** -- The color cyan.
- **static Color DARK_GRAY** -- The color dark gray.
- **static Color darkGray** -- The color dark gray.
- **static Color gray** -- The color gray.
- **static Color GRAY** -- The color gray.
- **static Color green** -- The color green.
- **static Color GREEN** -- The color green.
- **static Color LIGHT_GRAY** -- The color light gray.
- **static Color lightGray** -- The color light gray.
- **static Color magenta** -- The color magenta.
- **static Color MAGENTA** -- The color magenta.
- **static Color orange** -- The color orange.
- **static Color ORANGE** -- The color orange.
- **static Color pink** -- The color pink.
- **static Color PINK** -- The color pink.
- **static Color red** -- The color red.
- **static Color RED** -- The color red.
- **static Color white** -- The color white.
- **static Color WHITE** -- The color white.
- **static Color yellow** -- The color yellow.
- **static Color YELLOW** -- The color yellow.

Class Constructors

S.N.	Constructor & Description
1	**Color(ColorSpace cspace, float[] components, float alpha)** Creates a color in the specified ColorSpace with the color components specified in the float array and the specified alpha.
2	**Color(float r, float g, float b)** Creates an opaque sRGB color with the specified red, green, and blue values in the range (0.0 - 1.0).

3	**Color(float r, float g, float b, float a)** Creates an sRGB color with the specified red, green, blue, and alpha values in the range (0.0 - 1.0).
4	**Color(int rgb)** Creates an opaque sRGB color with the specified combined RGB value consisting of the red component in bits 16-23, the green component in bits 8-15, and the blue component in bits 0-7.
5	**Color(int rgba, boolean hasalpha)** Creates an sRGB color with the specified combined RGBA value consisting of the alpha component in bits 24-31, the red component in bits 16-23, the green component in bits 8-15, and the blue component in bits 0-7.
6	**Color(int r, int g, int b)** Creates an opaque sRGB color with the specified red, green, and blue values in the range (0 - 255).
7	**Color(int r, int g, int b, int a)** Creates an sRGB color with the specified red, green, blue, and alpha values in the range (0 - 255).

Class Methods

S.N.	Method & Description
1	**Color brighter()** Creates a new Color that is a brighter version of this Color.
2	**PaintContext createContext(ColorModel cm, Rectangle r, Rectangle2D r2d, AffineTransform xform, RenderingHints hints)** Creates and returns a PaintContext used to generate a solid color pattern.
3	**Color darker()** Creates a new Color that is a darker version of this Color.
4	**static Color decode(String nm)** Converts a String to an integer and returns the specified opaque Color.
5	**boolean equals(Object obj)** Determines whether another object is equal to this Color.
6	**int getAlpha()** Returns the alpha component in the range 0-255.
7	**int getBlue()** Returns the blue component in the range 0-255 in the default sRGB space.
8	**static Color getColor(String nm)** Finds a color in the system properties.
9	**static Color getColor(String nm, Color v)** Finds a color in the system properties.
10	**static Color getColor(String nm, int v)** Finds a color in the system properties.
11	**float[] getColorComponents(ColorSpace cspace, float[] compArray)** Returns a float array containing only the color components of the Color in the ColorSpace specified by the cspace parameter.

12	**float[] getColorComponents(float[] compArray)** Returns a float array containing only the color components of the Color, in the ColorSpace of the Color.
13	**ColorSpace getColorSpace()** Returns the ColorSpace of this Color.
14	**float[] getComponents(ColorSpace cspace, float[] compArray)** Returns a float array containing the color and alpha components of the Color, in the ColorSpace specified by the cspace parameter.
15	**float[] getComponents(float[] compArray)** Returns a float array containing the color and alpha components of the Color, in the ColorSpace of the Color.
16	**int getGreen()** Returns the green component in the range 0-255 in the default sRGB space.
17	**static Color getHSBColor(float h, float s, float b)** Creates a Color object based on the specified values for the HSB color model.
18	**int getRed()** Returns the red component in the range 0-255 in the default sRGB space.
19	**int getRGB()** Returns the RGB value representing the color in the default sRGB ColorModel.
20	**float[] getRGBColorComponents(float[] compArray)** Returns a float array containing only the color components of the Color, in the default sRGB color space.
21	**float[] getRGBComponents(float[] compArray)** Returns a float array containing the color and alpha components of the Color, as represented in the default sRGB color space.
22	**int getTransparency()** Returns the transparency mode for this Color.
23	**int hashCode()** Computes the hash code for this Color.
24	**static int HSBtoRGB(float hue, float saturation, float brightness)** Converts the components of a color, as specified by the HSB model, to an equivalent set of values for the default RGB model.
25	**static float[] RGBtoHSB(int r, int g, int b, float[] hsbvals)** Converts the components of a color, as specified by the default RGB model, to an equivalent set of values for hue, saturation, and brightness that are the three components of the HSB model.
26	**String toString()** Returns a string representation of this Color.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

Color Example

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

AWTGraphicsDemo.java

```
package com.tutorialspoint.gui;

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class AWTGraphicsDemo extends Frame {

   public AWTGraphicsDemo(){
      super("Java AWT Examples");
      prepareGUI();
   }

   public static void main(String[] args){
      AWTGraphicsDemo  awtGraphicsDemo = new AWTGraphicsDemo();
      awtGraphicsDemo.setVisible(true);
   }

   private void prepareGUI(){
      setSize(400,400);
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
   }

   @Override
```

```
   public void paint(Graphics g) {
      Graphics2D g2 = (Graphics2D)g;
      Font plainFont = new Font("Serif", Font.PLAIN, 24);
      g2.setFont(plainFont);
      g2.setColor(Color.red);
      g2.drawString("Welcome to TutorialsPoint", 50, 70);
      g2.setColor(Color.GRAY);
      g2.drawString("Welcome to TutorialsPoint", 50, 120);
   }
}
```

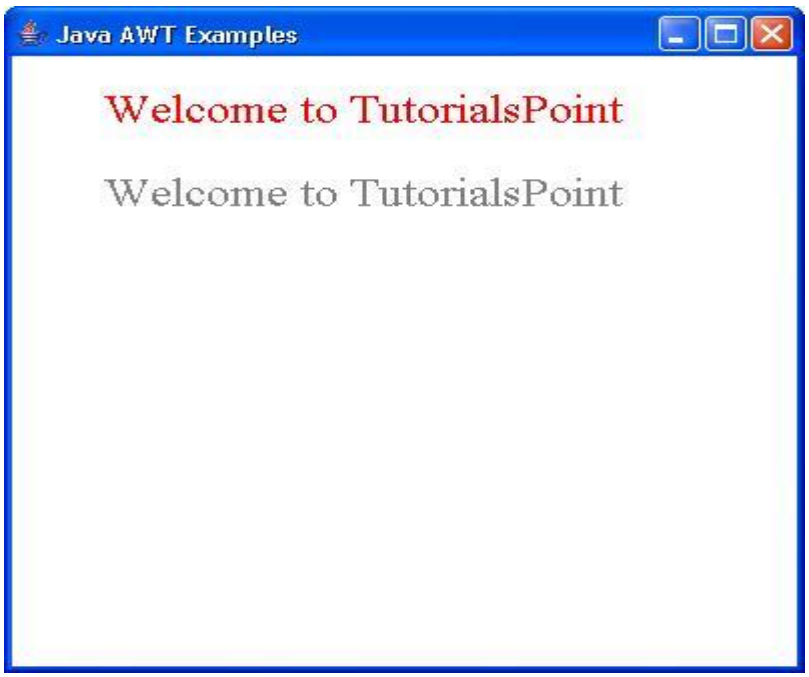
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command:

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

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

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

Output



AWT BasicStroke Class

The BasicStroke class states colors in the default sRGB color space or colors in the arbitrary color spaces identified by a ColorSpace.

Class Declaration

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

```
public class BasicStroke
   extends Object
      implements Stroke
```

Field

Following are the fields for **java.awt.geom.Arc2D** class:

- **static int CAP_BUTT** -- Ends unclosed subpaths and dash segments with no added decoration.
- **static int CAP_ROUND** -- Ends unclosed subpaths and dash segments with a round decoration that has a radius equal to half of the width of the pen.
- **static int CAP_SQUARE** -- Ends unclosed subpaths and dash segments with a square projection that extends beyond the end of the segment to a distance equal to half of the line width.
- **static int JOIN_BEVEL** -- Joins path segments by connecting the outer corners of their wide outlines with a straight segment.
- **static int JOIN_MITER** -- Joins path segments by extending their outside edges until they meet.
- **static int JOIN_ROUND** -- Joins path segments by rounding off the corner at a radius of half the line width.

Class Constructors

S.N.	Constructor & Description
1	**BasicStroke()** Constructs a new BasicStroke with defaults for all attributes.
2	**BasicStroke(float width)** Constructs a solid BasicStroke with the specified line width and with default values for the cap and join styles.
3	**BasicStroke(float width, int cap, int join)** Constructs a solid BasicStroke with the specified attributes.
4	**BasicStroke(float width, int cap, int join, float miterlimit)** Constructs a solid BasicStroke with the specified attributes.

5	**BasicStroke(float width, int cap, int join, float miterlimit, float[] dash, float dash_phase)** Constructs a new BasicStroke with the specified attributes.

Class Methods

S.N.	Method & Description
1	**Shape createStrokedShape(Shape s)** Returns a Shape whose interior defines the stroked outline of a specified Shape.
2	**boolean equals(Object obj)** Tests if a specified object is equal to this BasicStroke by first testing if it is a BasicStroke and then comparing its width, join, cap, miter limit, dash, and dash phase attributes with those of this BasicStroke.
3	**float[] getDashArray()** Returns the array representing the lengths of the dash segments.
4	**float getDashPhase()** Returns the current dash phase.
5	**int getEndCap()** Returns the end cap style.
6	**int getLineJoin()** Returns the line join style.
7	**float getLineWidth()** Returns the line width.
8	**float getMiterLimit()** Returns the limit of miter joins.
9	**int hashCode()** Returns the hashcode for this stroke.

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

BasicStroke Example

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

AWTGraphicsDemo.java

```
package com.tutorialspoint.gui;


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

```
public class AWTGraphicsDemo extends Frame {

   public AWTGraphicsDemo(){
      super("Java AWT Examples");
      prepareGUI();
   }

   public static void main(String[] args){
      AWTGraphicsDemo  awtGraphicsDemo = new AWTGraphicsDemo();
      awtGraphicsDemo.setVisible(true);
   }

   private void prepareGUI(){
      setSize(400,400);
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
   }

   @Override
   public void paint(Graphics g) {
      Graphics2D g2 = (Graphics2D)g;
      g2.setStroke(new BasicStroke(3.0f));
      g2.setPaint(Color.blue);

      Rectangle2D shape = new Rectangle2D.Float();
      shape.setFrame(100, 150, 200,100);
      g2.draw(shape);

      Rectangle2D shape1 = new Rectangle2D.Float();
```

```
      shape1.setFrame(110, 160, 180,80);
      g2.setStroke(new BasicStroke(1.0f));

      g2.draw(shape1);
      Font plainFont = new Font("Serif", Font.PLAIN, 24);
      g2.setFont(plainFont);
      g2.setColor(Color.DARK_GRAY);
      g2.drawString("TutorialsPoint", 130, 200);
   }
}
```

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

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

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

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

Output