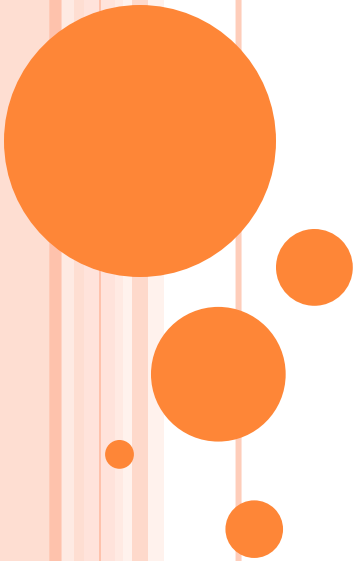
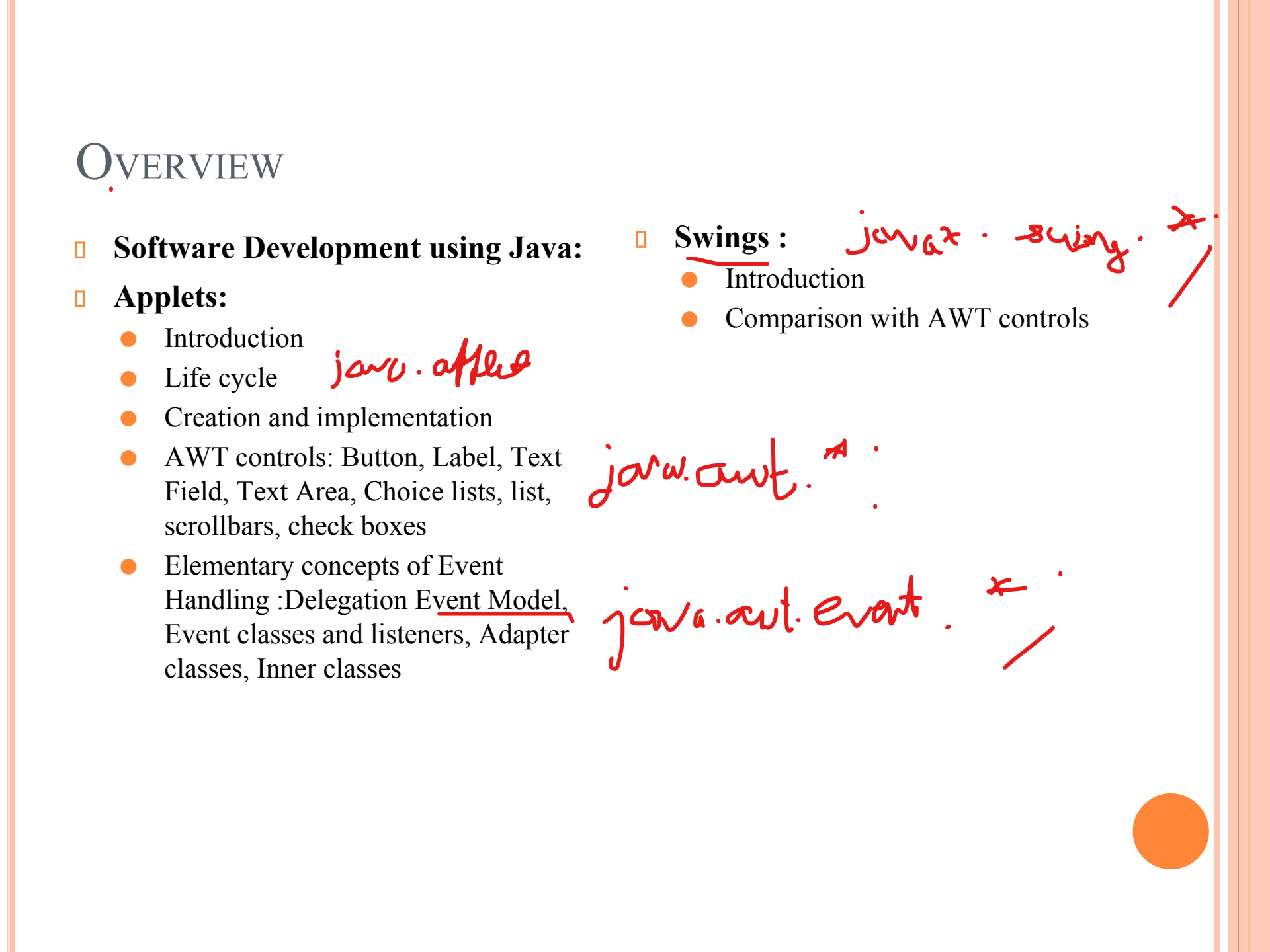


UNIT-III JAVA

Applets and Swings

Overview

- **Software Development using Java:**
- **Applets:**
  - Introduction
  - Life cycle
  - Creation and implementation
  - AWT controls: Button, Label, Text Field, Text Area, Choice lists, list, scrollbars, check boxes
  - Elementary concepts of Event Handling :Delegation Event Model, Event classes and listeners, Adapter classes, Inner classes
- **Swings :**
  - Introduction
  - Comparison with AWT controls



Applets

- Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.
- Applets are small java programs that are primarily used in internet computing. They can be transported over the internet from one computer to another and can run using the "AppletViewer" or any web browser that support java.
- An applet like an application can do many things for us. It can perform arithmetic operation, display graphics, play sounds, accept user input, create animation and play interactive games.

Contd…

Advantage of Applets:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os etc.

Drawback of Applets:

- Plugin is required at client browser to execute applet.

Local and Remote Applets

There are two types of applet:

- Local applet
- Remote Applet

Local Applet: An applet developed locally and stored in a local system is known as local applet. When a web page is trying to find local applet, it does not need to use internet and therefore the local system does not require internet connection. It simply searches directories in the local system and locates and loads the specified applet.

Remote Applet: A remote applet is that which is developed by someone else and stored on remote computer connected to internet. If our system is connected to internet, we can download the remote applet onto our system via internet and run it. In order to download remote applet we must know the applet's URL that is applets address on the net.

How applets differ from applications?

- Both applet and applications are java programs but there are significant differences between them. Applets are not full featured application programs. They are usually written to accomplish small task. Since they are usually designed for use on internet, they impose certain limitations and restrictions in their design.
- Applets do not use the main() method for initiating the execution of the code. Rather Applets when loaded automatically call certain methods of applet class to start and execute the applet code.
- Unlike stand alone application , applets cannot run independently. They run inside a web page.
- Applets can not read from or write to the files in the local computer.

Contd…

- Applet cannot communicate with other servers on the network.
- It cannot make network connections except to the host that it came from.
- Applets can not run any program from the local computer.
- Applets are restricted from using libraries from other languages such as C or C++.
- Windows that an applet brings up look different than windows that an application brings up.
- All these limitations are placed in the interest of security of systems. These restrictions ensure that an applet cannot do any damage to the local system.

Java Packages required for Applets

- It is essential that our applet code uses the services of two classes: **"Applet"** and **"Graphics"** from java class library.
- **Applet**: class is contained in the java.applet package provides life and behavior to the applet through various methods. Applet class maintains the lifecycle of an applet.
- **Graphics**: when applet is called, For displaying the result of applet code on the screen, paint() method is used , paint() method requires object of Graphics class, which is available in java.awt package.
- Example:

```
Import java.applet.*;
Import java.awt.*;
Public class appletclassname extends Applet
{
// applet functions code
}
```

First Applet Program

```
import java.awt.*;
import java.applet.*;
public class SimpleApplet extends Applet {
public void paint(Graphics g) {
g.drawString("MY FIRST Applet CODE", 120,120);
}
}
```

- This applet begins with two **import** statements. The first imports the Abstract Window Toolkit (AWT) classes.
- The second **import** statement imports the **applet** package, which contains the class **Applet**.
- The next line in the program declares the class **SimpleApplet**. This class must be declared as **public**, because it will be accessed by code that is outside the program.

Contd…

- Inside **SimpleApplet**, **paint( )** is declared. This method is defined by the AWT and must be overridden by the applet.
- **paint( )** is called each time that the applet must redisplay its output.
- The **paint( )** method has one parameter of type **Graphics**. This parameter contains the graphics context, which describes the graphics environment in which the applet is running.
- Inside **paint( )** is a call to **drawString( )**, which is a member of the **Graphics** class.
- This method outputs a string beginning at the specified X,Y location. It has the following general form:

  void drawString(String *message*, int *x*, int *y*)

- Here, *message* is the string to be output beginning at *x*,*y*.

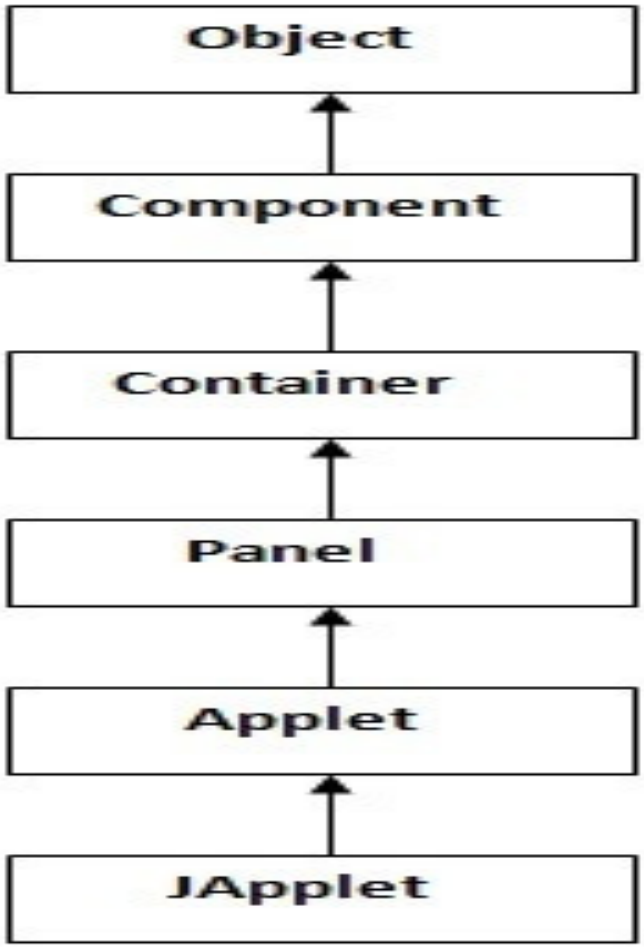
Contd…

- In a Java window, the upper-left corner is location 0,0. The call to **drawString( )** in the applet causes the message "A Simple Applet" to be displayed beginning at location 20,20.
- Notice that the applet does not have a **main( )** method. Unlike Java programs, applets do not begin execution at **main( )**.
- To execute an applet in a Web browser, you need to write a short HTML text file that contains the appropriate APPLET tag. Here is the HTML file that executes **SimpleApplet**:

```
<applet code="SimpleApplet" width=200 height=60>
</applet>
```

- The **width** and **height** statements specify the dimensions of the display area used by the applet.
- **Note**: Save this file by the name of SimpleApplet.java
- And run this file by appletviewer SimpleApplet.java.

Hierarchy of Applet



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

Lifecycle of Java Applet

- Applet is initialized.
- Applet is started.
- Applet is painted.
- Applet is stopped.
- Applet is destroyed.

- A java applet has following states:
  - Born [init()]
  - Running State [start()]
  - Idle state [stop()]
  - Dead or destroyed [destroy()]

CONTD…

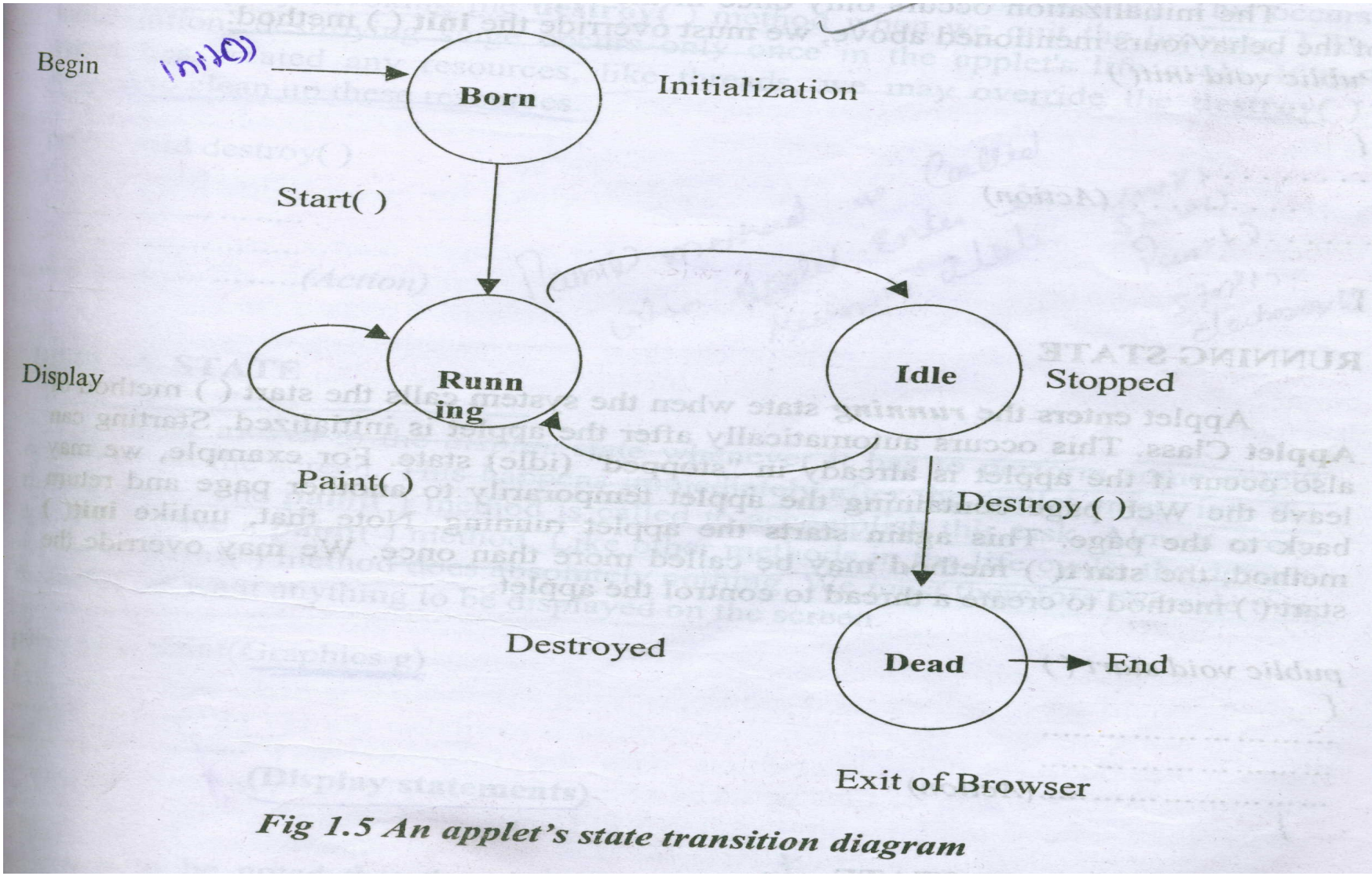


Fig 1.5 An applet's state transition diagram

CONTD…

- **Initialization State**: applet enters the initialization state when it is first loaded. This is achieved by calling the init() methods of applet class. The applet is born at this stage. In general, the init method should contain the code that you would normally put into a constructor.
- We can do the following at this stage create objects needed by the applet
  - Set up initial values
  - Load images or fonts
  - Set up colors
- The initialization occurs only once in the life cycle of the applet. to provide any of the above mentioned facility we must overwrite init() method.

```
Public void init()
{
// code for init method
}
```

CONTD…

- **Running Stage:** Applet enters the running stage when the system calls the start() method of Applet class. start() method is called after init(). This occurs automatically after the applet is initialized . Starting can also occur if the applet is already "stopped " idle state. start() method can be called more than once in an applet life cycle. Start() method is called every time when applet receives focus.
- We can overwrite the start() method.
- Public void start()
- {
- // code for start method
- }

- **Idle or stopped state:** An applet becomes idle when it is stopped from running . Stopping occurs automatically when we leave the page containing the currently running applet. We can also do so by calling the stop() method explicitly. If we use the thread to run the applet , then we must use stop() method to terminate the thread.
- We can achieve this by overriding the stop() method. You can restart them with start() method. The stop method should suspend the applet's execution, so that it doesn't take up system resources when the user isn't viewing the applet's page.
- Public void stop()
- {
- // code for stop method
- }

- **Dead state**: an applet is said to be dead when it is removed from the memory. This occurs automatically by invoking the destroy() method. When we quit the browser. Like applet has created any resource, like threads we may override the destroy() method to clean up these resources. The stop method is always called before destroy().

- Public void destroy()
- {
- // code for destroy() method
- }

- **Display state** : applet moves to the display state whenever it has to perform output operations on the screen. This happens immediately after the applet enters into the running state. The paint() method is called to accomplish this task. Almost every applet will have a paint() method. The default version of the paint() method does absolutely nothing. We must therefore override this method if we want anything to be displayed on the screen.
- Public void paint (Graphics g)
- {
- }

- It is to be noted that the display state is not considered as a part of life cycle. In fact , the paint() method is not defined in the Applet class. It is inherited from the Component class, a super class of Applet.

Lifecycle methods for Applet:

- For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.
  - **public void init():** is used to initialized the Applet. It is invoked only once.
  - **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
  - **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
  - **public void destroy():** is used to destroy the Applet. It is invoked only once.

How to run an Applet?

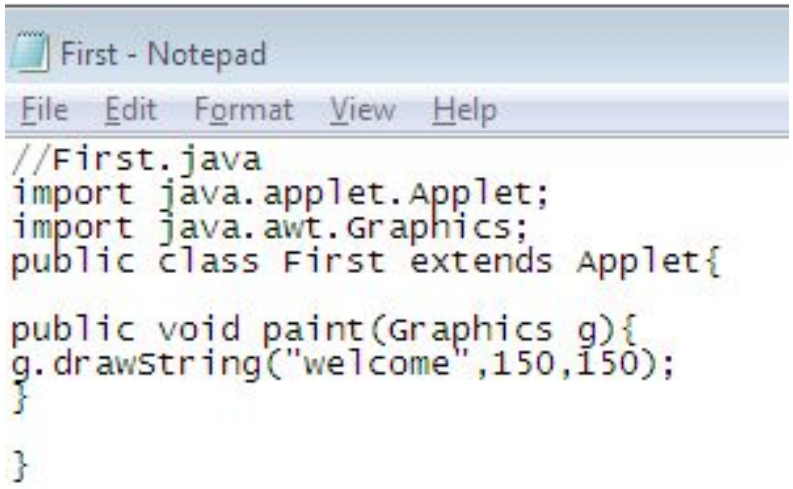
- There are two ways to run an applet
  - By html file.
  - By appletViewer tool (for testing purpose).

Simple example of Applet by html file:

- To execute the applet by html file, create an applet and compile it.
- After that create an html file and place the applet code in html file.
- Now run the html file.
- Let us understand with an example-

EXAMPLE-

- **Step-1**: Create .java file with following coding & save it by the name of First.java



CONTD..

- **Step-2** Create an html file with following code & then click on this html file to view the applet.

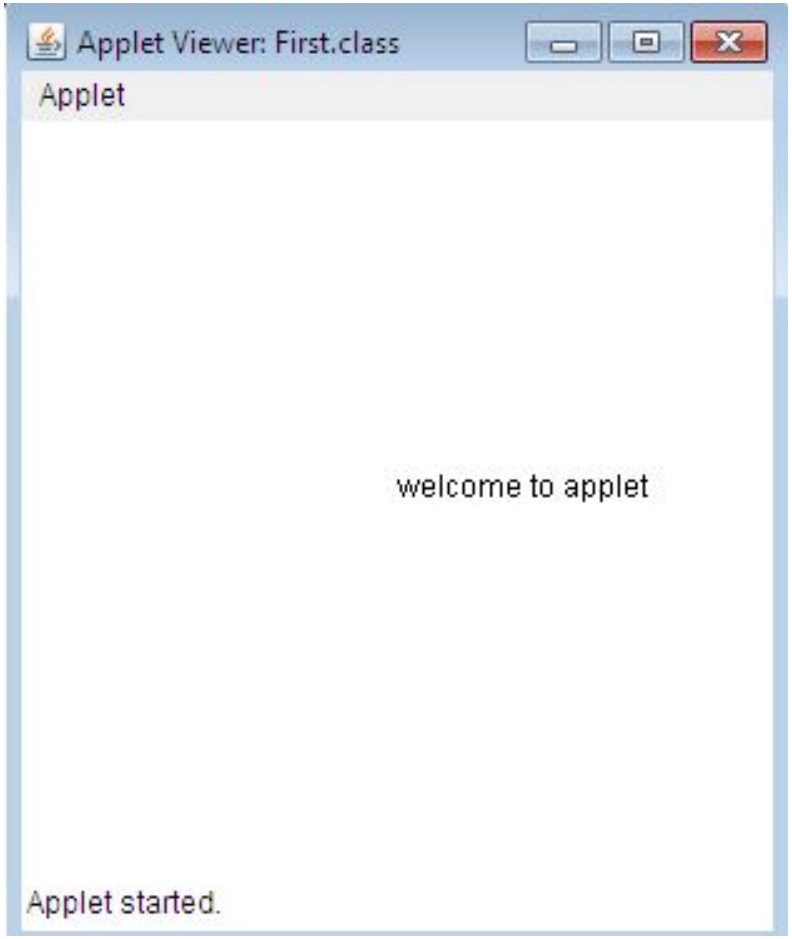
Firstapplet - Notepad

File Edit Format View Help

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

Step-3 Compile this java file using command prompt & view html file using appletviewer

OUTPUT-


Applet Viewer: First.class
Applet
welcome to applet
Applet started.

Simple example of Applet by appletviewer tool:

- To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it.
- After that run it by: appletviewer First.java.
- Now Html file is not required but it is for testing purpose only.
- Lets see an example for better understanding

Example

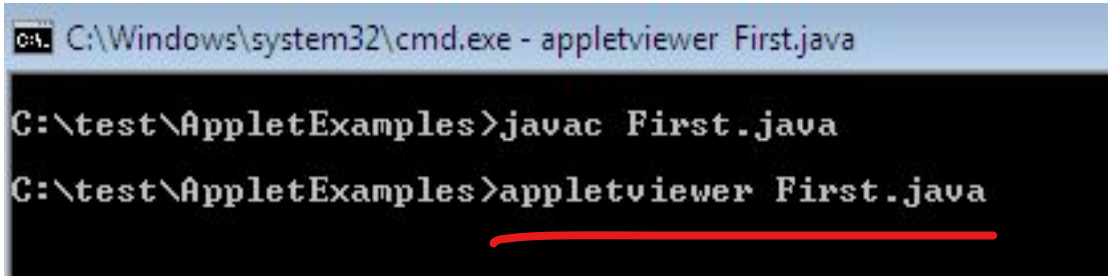
- **Step-1** Create .java file with following coding

First - Notepad

File Edit Format View Help

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{

public void paint(Graphics g){
g.drawString("welcome to applet",150,150);
}

}
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

Step-2: To execute the applet by appletviewer tool, write in command prompt:



Displaying Graphics in Applet

- java.awt.Graphics class provides many methods for graphics programming.
- **Commonly used methods of Graphics class:**
  - **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
  - **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
  - **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
  - **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
  - **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.

Contd..

- **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
- **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
- **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
- **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
- **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
- **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Example of Graphics in applet:

GraphicsDemo - Notepad

File Edit Format View Help

```
import java.applet.Applet;
import java.awt.*;

public class GraphicsDemo extends Applet{

public void paint(Graphics g){
g.setColor(Color.red);
g.drawString("Welcome",50, 50);
g.drawLine(20,30,20,300);
g.drawRect(70,100,30,30);
g.fillRect(170,100,30,30);
g.drawOval(70,200,30,30);

g.setColor(Color.pink);
g.fillOval(170,200,30,30);
g.drawArc(90,150,30,30,30,270);
g.fillArc(270,150,30,30,0,180);

}
}
```

GraphicsDemo.java

CONTD…

myapplet - Notepad

File Edit Format View Help

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

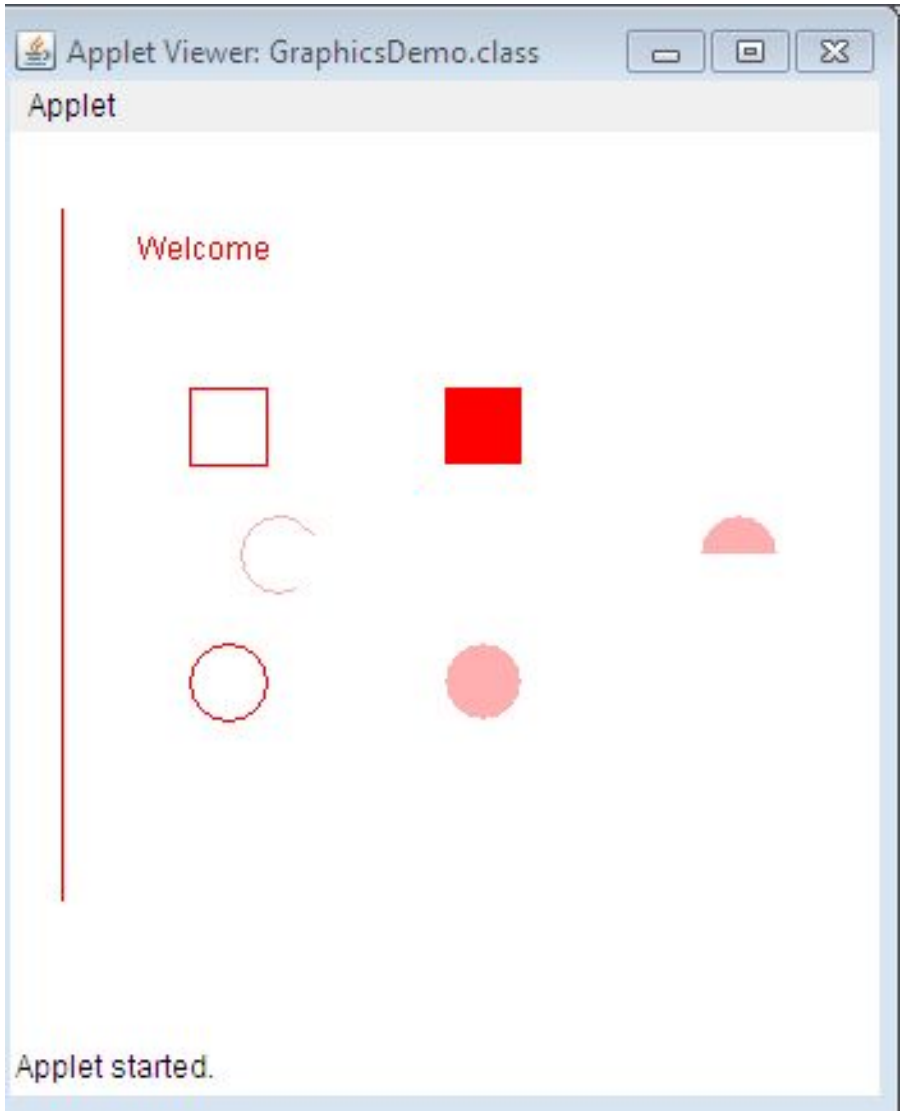
myapplet.html

C:\Windows\system32\cmd.exe

```
C:\test\AppletExamples>javac GraphicsDemo.java

C:\test\AppletExamples>appletviewer myapplet.html

C:\test\AppletExamples>
```

Compilation on cmd

OUTPUT-


Applet Viewer: GraphicsDemo.class
Applet
Welcome
Applet started.

Simple Applet Display Methods

- To set the background color of an applet's window, use setBackground( ). To set the foreground color (the color in which text is shown, for example), use setForeground( ). These methods are defined by Component, and they have the following general forms:

  void setBackground(Color newColor)

  void setForeground(Color newColor)

- Here, newColor specifies the new color. The class Color defines the constants shown here that can be used to specify colors:

Contd…

- Color.black
- Color.magenta
- Color.blue
- Color.orange
- Color.cyan
- Color.pink
- Color.darkGray
- Color.red
- Color.gray
- Color.white
- Color.green
- Color.yellow
- Color.lightGray
- For example, this sets the background color to green and the text color to red: setBackground(Color.green);
- setForeground(Color.red);

Applet Tag

- The applet tag is written within the body tag of an HTML document
- <applet
- Code = “ Name of the .class file”
- Codebase =“ path of .class file”
- Height= “maximum height of applet in pixel”
- Width = “maximum width of applet in pixel”
- Vspace = “vertical space between applet and rest of the HTML document”
- HSpace = “Horizontal space between the applet and the rest of the HTML document”
- Align = “alignment of the applet with respect to the rest of the web page”
- ALT= “alternative text to be displayed if the browser does not support applets”
- >
- </applet>

Contd…

- Applet tag has three (code, height, width) mandatory attributes rest all are optional attributes.

- **Types of applet Tag-**
- **Internal**: when applet tag is included in .java file within multiline comment entry. For executing applet following command is used:
- Appletviewer filename.java
- **External**: when applet tag is written in separate html file. For executing applet following command is used:
- Appletviewer filename.html

EXAMPLE-

Sample - Notepad

File Edit Format View Help

```
import java.awt.*;
import java.applet.*;
/*
<applet code="Sample" width=300 height=50>
</applet>
*/
public class Sample extends Applet{
String msg;
public void init() {
setBackground(Color.cyan);
setForeground(Color.red);
msg = "Inside init( ) --";
}
public void start() {
msg += " Inside start( ) --";
}
public void paint(Graphics g) {
msg += " Inside paint( ).";
g.drawString(msg, 10, 30);
}
}
```

Sample.java

C:\Windows\system32\cmd.exe

```
C:\test\AppletExamples>javac Sample.java

C:\test\AppletExamples>appletviewer Sample.java
```

Compilation statement as applet tag is inside .java file so no need to create .html file

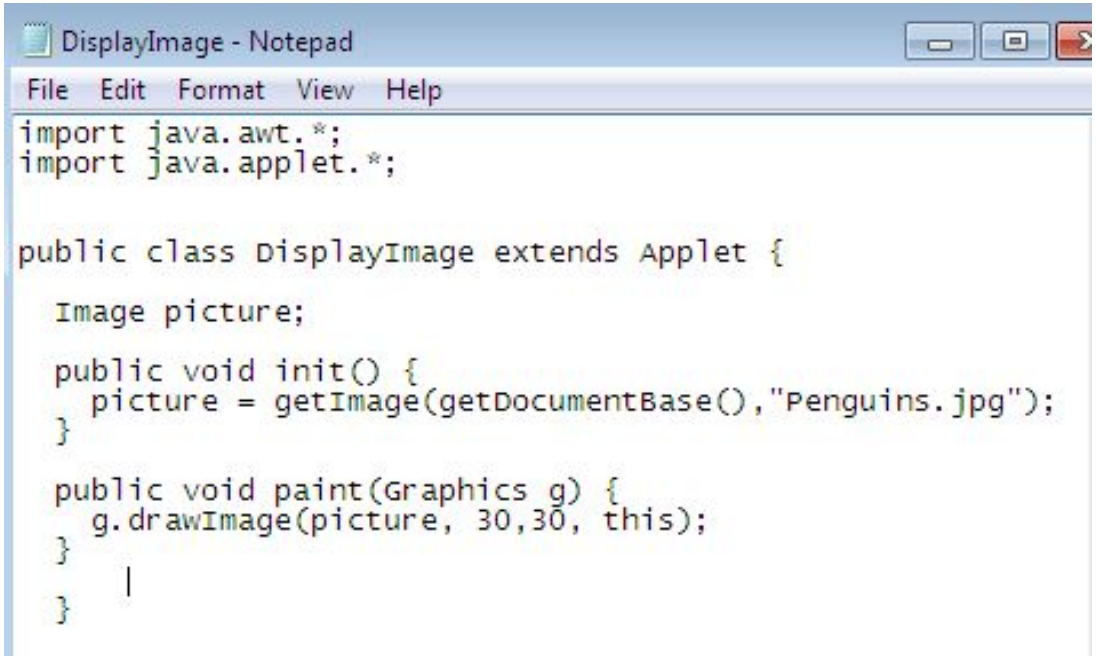
OUTPUT


Applet Viewer: Sample
Applet
Inside init( ) -- Inside start( ) -- Inside paint( ). Inside paint( ). Inside paint( ). Inside pain
Applet started.

Displaying Image in Applet

- Applet is mostly used in games and animation. For this purpose image is required to be displayed.
- The java.awt.Graphics class provide a method drawImage() to display the image.
- **Syntax of drawImage() method:**
  - **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
- **How to get the object of Image:**
- The java.applet.Applet class provides getImage() method that returns the object of Image.
- Syntax:
  - **public** Image getImage(URL u, String image){}

Example-



DisplayImage.java

In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.

CONTD…

displayapplet.html

displayapplet - Notepad

File Edit Format View Help

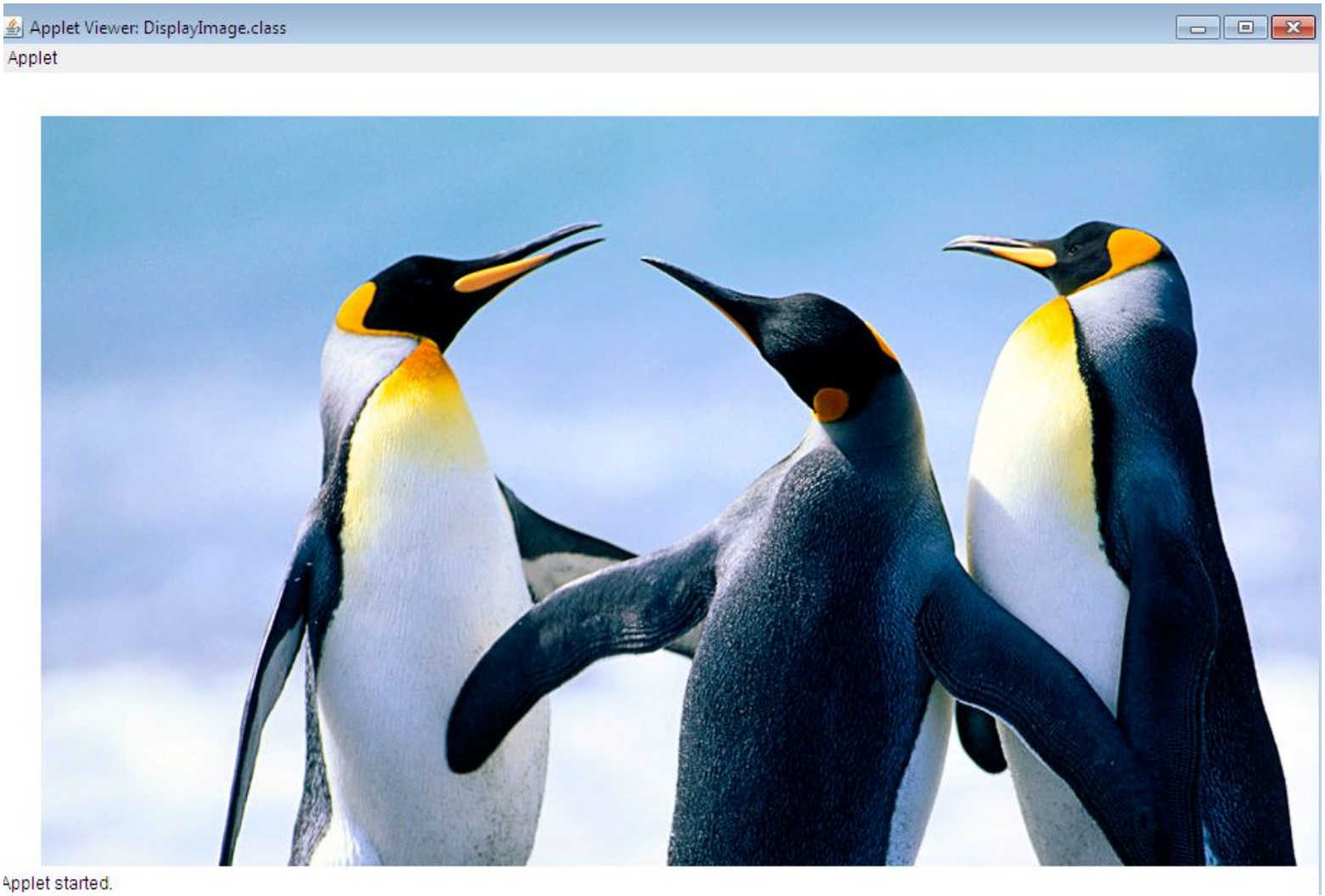
```
<html>
<body>
<applet code="DisplayImage.class" width="300" height="300">
</applet>
</body>
</html>
```

Compiling statement

C:\Windows\system32\cmd.exe

```
C:\test\AppletExamples>javac DisplayImage.java

C:\test\AppletExamples>appletviewer displayapplet.html
```

Output-



When to write Applet Vs Application

- In the early days of Java, one of the critical advantages that Java applets had over Java applications was that applets could be easily deployed over the web while Java applications required a more cumbersome installation process.
- Additionally, since applets are downloaded from the internet, by default they have to run in a restricted security environment, called the "sandbox", to ensure they don't perform any destructive operations on the user's computer, such as reading/writing to the file system.
- However, the introduction of Java Web Start has made it possible for Java applications to also be easily deployed over the web, as well as run in a secure environment

Contd…

- This means that the predominant difference between a Java applet and a Java application is that an applet runs in the context of a web browser, being typically embedded within an html page, while a Java application runs standalone, outside the browser.
- Thus, applets are particularly well suited for providing functions in a web page which require more interactivity or animation than HTML can provide, such as a graphical game, complex editing, or interactive data visualization. The end user is able to access the functionality without leaving the browser.

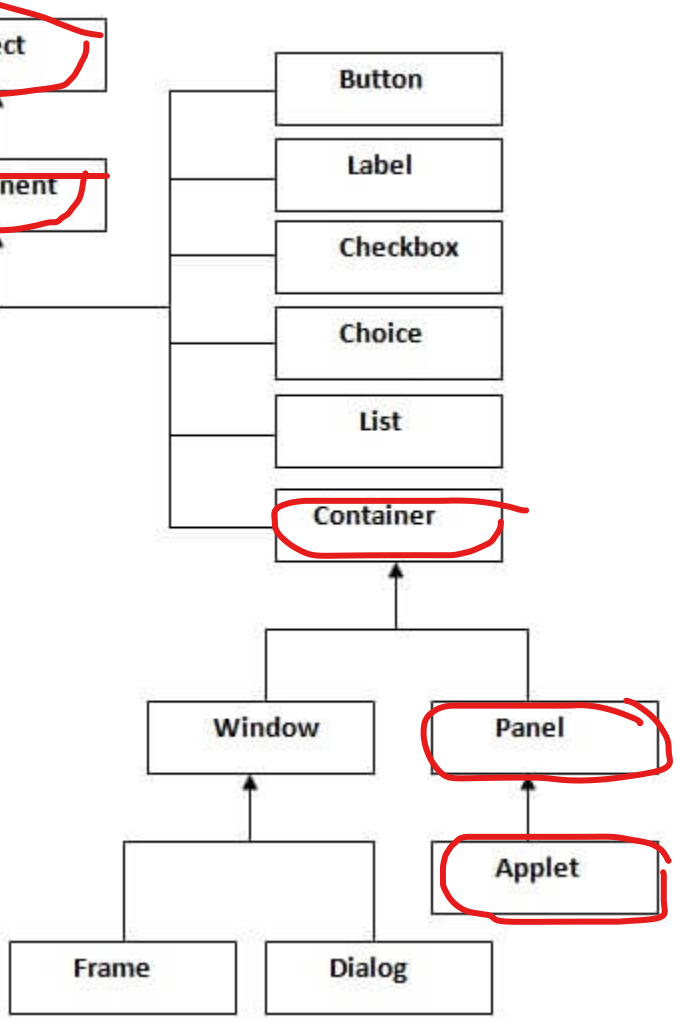
JAVA AWT (Abstract Window Toolkit)

Java AWT

- **Java AWT** (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system.
- AWT is heavyweight i.e. its components are using the resources of OS.
- The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy

- The hierarchy of Java AWT classes are given below.



Explanation-

Container

- The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window

- The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

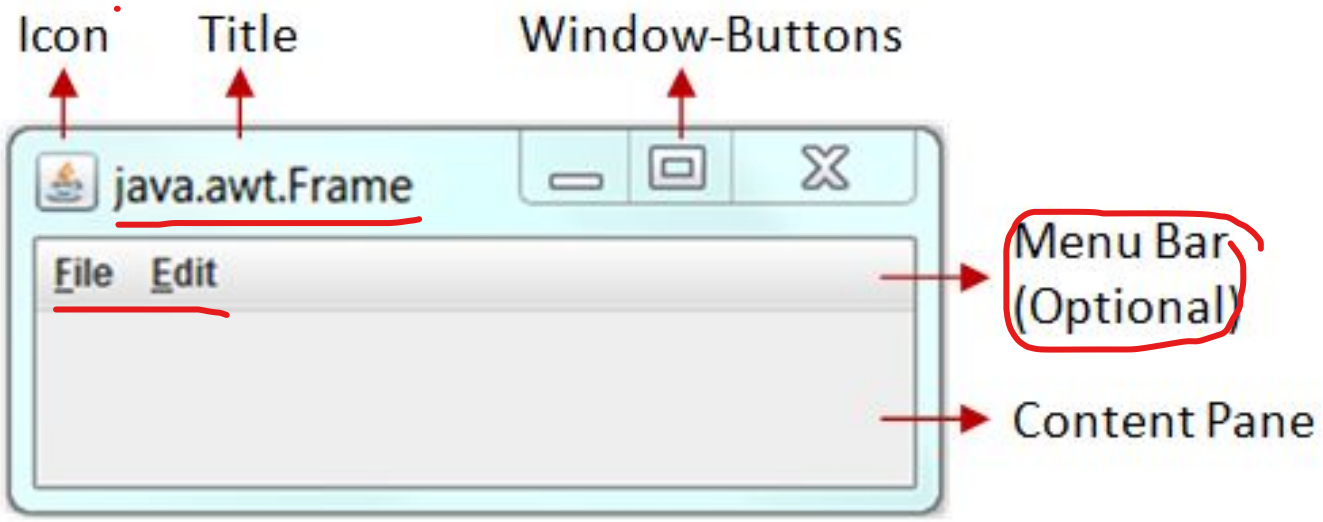
Panel

- The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame

- The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

EXAMPLE-


Icon
Title
Window-Buttons
java.awt.Frame
File Edit
Menu Bar
(Optional)
Content Pane

AWT PACKAGE

- **Working with AWT Classes**
- The AWT classes are contained in the java.awt package.
- Following are some of the AWT Classes

Class	Description
Canvas	A blank, semantics-free window.
CardLayout	The card layout manager. Card layouts emulate index cards. Only the one on top is showing.
Checkbox	Creates a check box control.
CheckboxGroup	Creates a group of check box controls.
CheckboxMenuItem	Creates an on/off menu item.
Choice	Creates a pop-up list.
Color	Manages colors in a portable, platform-independent fashion.
Component	An abstract superclass for various AWT components.
Container	A subclass of **Component** that can hold other components.
Cursor	Encapsulates a bitmapped cursor.
Dialog	Creates a dialog window.
Dimension	Specifies the dimensions of an object. The width is stored in **width**, and the height is stored in **height**.
Event	Encapsulates events.
EventQueue	Queues events.
FileDialog	Creates a window from which a file can be selected.
FlowLayout	The flow layout manager. Flow layout positions components left to right, top to bottom.
Font	Encapsulates a type font.
FontMetrics	Encapsulates various information related to a font. This information helps you display text in a window.
Frame	Creates a standard window that has a title bar, resize corners, and a menu bar.
Graphics	Encapsulates the graphics context. This context is used by the various output methods to display output in a window.
GraphicsDevice	Describes a graphics device such as a screen or printer.

CONTD….

Class	Description
GraphicsEnvironment	Describes the collection of available **Font** and **GraphicsDevice** objects.
GridBagConstraints	Defines various constraints relating to the **GridBagLayout** class.
GridBagLayout	The grid bag layout manager. Grid bag layout displays components subject to the constraints specified by **GridBagConstraints**.
GridLayout	The grid layout manager. Grid layout displays components in a two-dimensional grid.
Image	Encapsulates graphical images.
Insets	Encapsulates the borders of a container.
Label	Creates a label that displays a string.
List	Creates a list from which the user can choose. Similar to the standard Windows list box.
MediaTracker	Manages media objects.
Menu	Creates a pull-down menu.
MenuBar	Creates a menu bar.
MenuComponent	An abstract class implemented by various menu classes.
MenuItem	Creates a menu item.
MenuShortcut	Encapsulates a keyboard shortcut for a menu item.
Panel	The simplest concrete subclass of **Container**.
Point	Encapsulates a Cartesian coordinate pair, stored in **x** and **y**.
Polygon	Encapsulates a polygon.
PopupMenu	Encapsulates a pop-up menu.
PrintJob	An abstract class that represents a print job.
Rectangle	Encapsulates a rectangle.
Robot	Supports automated testing of AWT- based applications. (Added by Java 2, v1.3)
Scrollbar	Creates a scroll bar control.

CONTD…

ScrollPane	A container that provides horizontal and/or vertical scroll bars for another component.
SystemColor	Contains the colors of GUI widgets such as windows, scroll bars, text, and others.
TextArea	Creates a multiline edit control.
TextComponent	A superclass for **TextArea** and **TextField**.
TextField	Creates a single-line edit control.
Toolkit	Abstract class implemented by the AWT.
Window	Creates a window with no frame, no menu bar, and no title.

Table 21-1. *Some AWT Classes* (continued)

Useful Methods of Component class

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

Java AWT Example

- To create simple awt example, you need a frame.
- There are two ways to create a frame in AWT:
  - By extending Frame class (inheritance)
  - By creating the object of Frame class (association)

AWT Example by Inheritance

- Let's see a simple example of AWT where we are inheriting Frame class.
- Here, we are showing Button component on the Frame.

FirstFrame.java

```
import java.awt.*;
class FirstFrame extends Frame{
FirstFrame(){
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position
add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
setLayout(null);//no layout manager
setVisible(true);//now frame will be visible, by default not visible
}
public static void main(String args[]){
FirstFrame f=new FirstFrame();
}}
```

The setBounds(int xaxis, int yaxis, int width, int height) method is used in the above example that sets the position of the awt button.

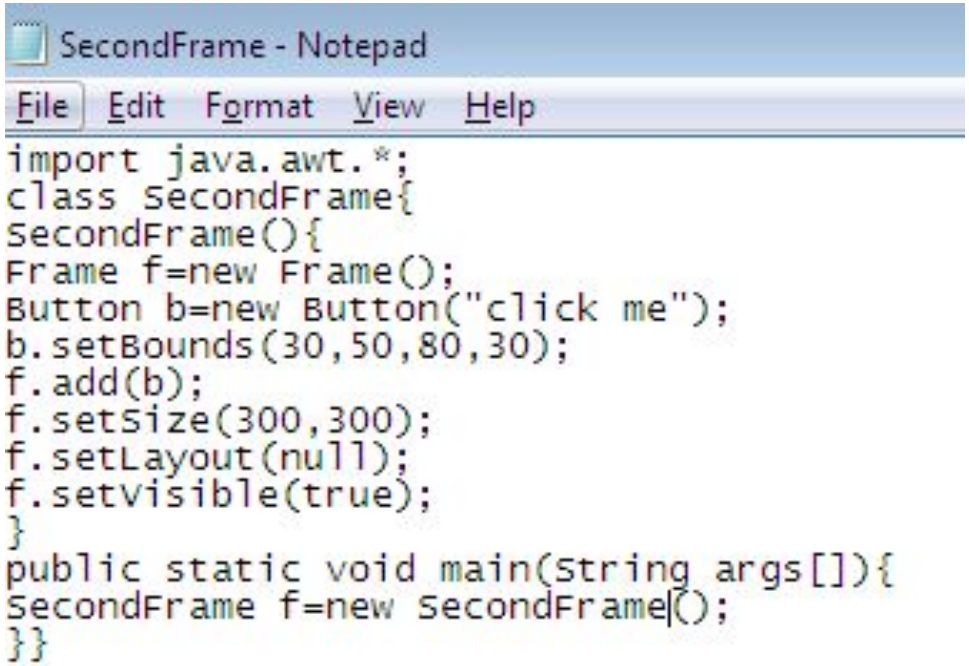
CONTD…

Compilation statement

Output

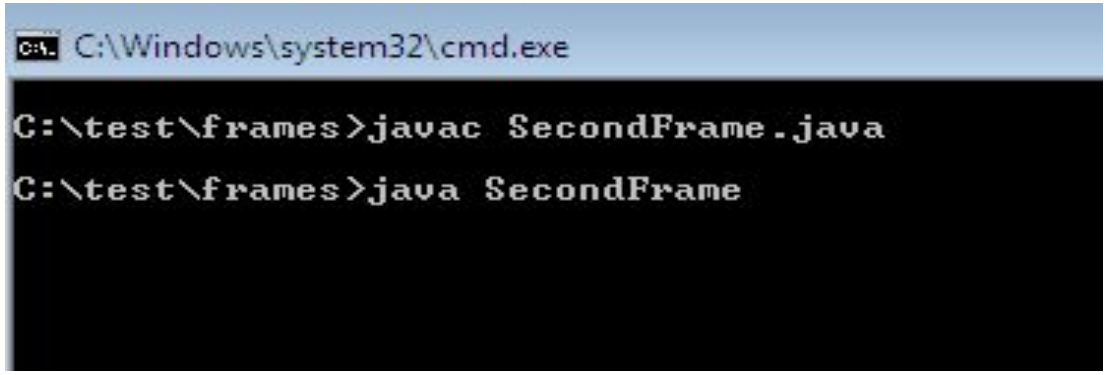
AWT Example by Association

- Let's see a simple example of AWT where we are creating instance of Frame class.
- Here, we are showing Button component on the Frame.



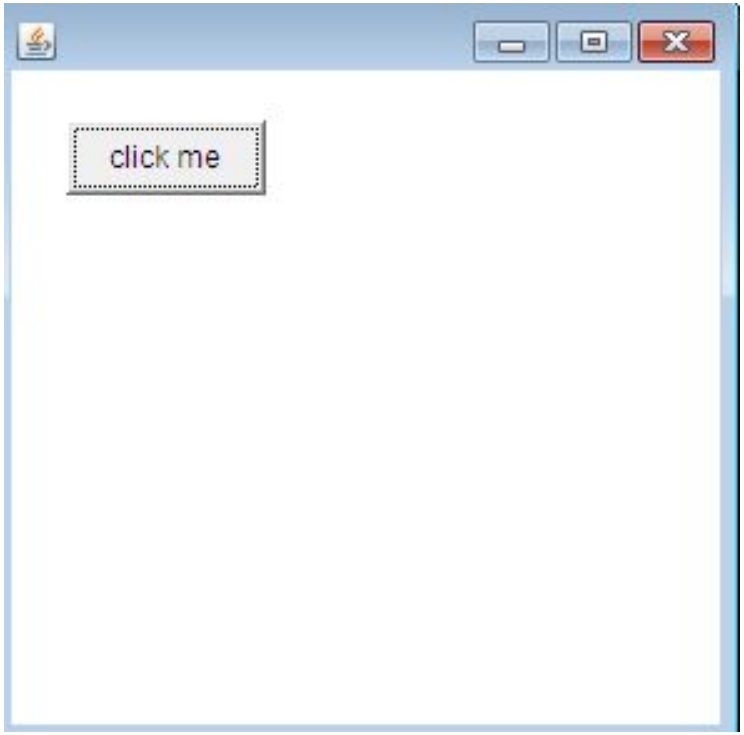
SecondFrame.java

CONTD…



Compilation statement

Output



AWT Controls

- There are several AWT controls available. Some of them are given below:
  - **Labels**
  - **Push buttons**
  - **Check boxes**
  - **Radio button/checkbox group/option button**
  - **Choice lists/combo box**
  - **Lists**
  - **Canvas**
  - **Scroll bars**
  - **Text Field**
  - **Text Area**

Adding & Removing Controls

- To include a control in a window, you must add it to the window. To do this, you must first create an instance of the desired control and then add it to a window by calling add( ) which is defined by Container.
- The add( ) method has several forms.
  - Component add(Component compObj)
- Here, compObj is an instance of the control that you want to add. A reference to compObj is returned. Once a control has been added, it will automatically be visible whenever its parent window is displayed.
  Example

```
Button b= new Button("click me")
f.add(b);
```

- Sometimes you will want to remove a control from a window when the control is no longer needed. To do this, call remove( ). This method is also defined by Container. It has this general form:
  - void remove(Component obj)
- Here, obj is a reference to the control you want to remove.

Labels

- A *label* is an object of type Label, and it contains a string, which it displays. A label displays a single line in Read- only text. Text can be changed at application, user can not edit directly.Label defines the following constructors:
  - Label( )
  - Label(String *str*)
  - Label(String *str*, int alignment)
- The first version creates a blank label.
- The second version creates a label that contains the string specified by *str*. This string is left-justified.
- The third version creates a label that contains the string specified by *str* using the alignment specified . The value of alignment must be one of these three constants: Label.LEFT, Label.RIGHT, or Label.CENTER.

EXAMPLE

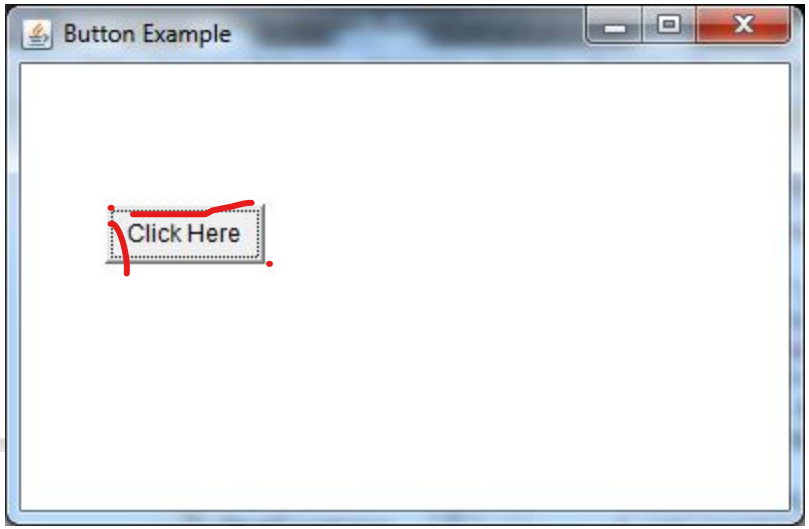
```
import java.awt.*;
class LabelExample{
public static void main(String args[]){
    Frame f= new Frame("Label Example");
    Label l,2;
    l=new Label("First Label.");
    1.setBounds(50,100, 100,30);
    2=new Label("Second Label.");
    2.setBounds(50,150, 100,30);
    f.add(1); f.add(2);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Push Buttons

- A button is a component that has a label and can respond when pressed. It has the following constructors:
- Button() – construct a button with no label
- Button(String label) – construct a button with specified label

Program to add a button on a frame

```
import java.awt.*;
public class ButtonExample {
public static void main(String[] args) {
    Frame f=new Frame("Button Example");
    Button b=new Button("Click Here");
    b.setBounds(50,100,80,30);
    f.add(b);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

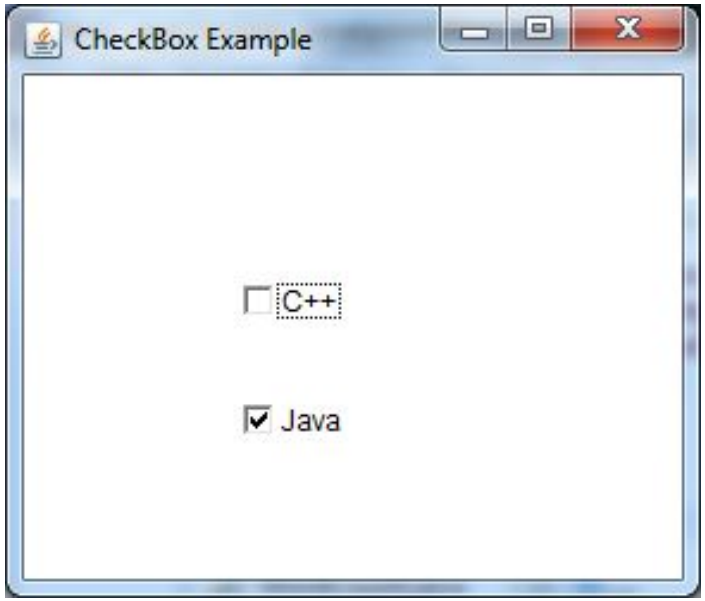


Check Boxes

- The checkbox class is used to create a labeled checkbox. It has two parts- a caption and state.
- The caption is text and represents the label of the control and the state is the boolean value.
- By default the state is false which means the check box is unchecked.
- The basic form of the checkbox constructors are:
- Checkbox(String s)
- Checkbox(String s, initial state)

Program to add checkbox on a frame

```
import java.awt.*;
public class CheckboxExample
{
    CheckboxExample(){
      Frame f= new Frame("Checkbox Example");
       Checkbox checkbox1 = new Checkbox("C++");
       checkbox1.setBounds(100,100, 50,50);
       Checkbox checkbox2 = new Checkbox("Java", true);
       checkbox2.setBounds(100,150, 50,50);
       f.add(checkbox1);
       f.add(checkbox2);
       f.setSize(400,400);
       f.setLayout(null);
       f.setVisible(true);
    }
public static void main(String args[])
{
   new CheckboxExample();
}
}
```



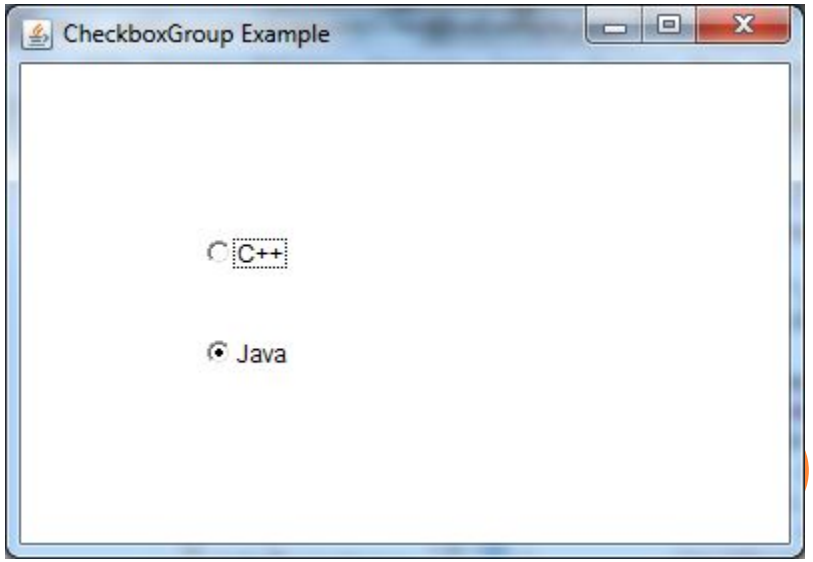
Radio button/checkbox group/option button

- The Checkbox group is also called as option button or radio button.
- Checked boxes are grouped together into a check box group. So only one member of a check box group is selected at a time.
- The difference between checkbox and checkbox group is that only one choice can be selected from a checkbox group wherever more than choice can be selected from a checkbox.

- **Note**: CheckboxGroup enables you to create radio buttons in AWT. There is no special control for creating radio buttons in AWT.

Program-

```
import java.awt.*;
public class CheckboxGroupExample
{
      CheckboxGroupExample(){
      Frame f= new Frame("CheckboxGroup Example");
       CheckboxGroup cbg = new CheckboxGroup();
       Checkbox checkBox1 = new Checkbox("C++", cbg, false);
       checkBox1.setBounds(100,100, 50,50);
       Checkbox checkBox2 = new Checkbox("Java", cbg, true);
       checkBox2.setBounds(100,150, 50,50);
       f.add(checkBox1);
       f.add(checkBox2);
       f.setSize(400,400);
       f.setLayout(null);
       f.setVisible(true);
    }
public static void main(String args[])
{
   new CheckboxGroupExample();
}
}
```

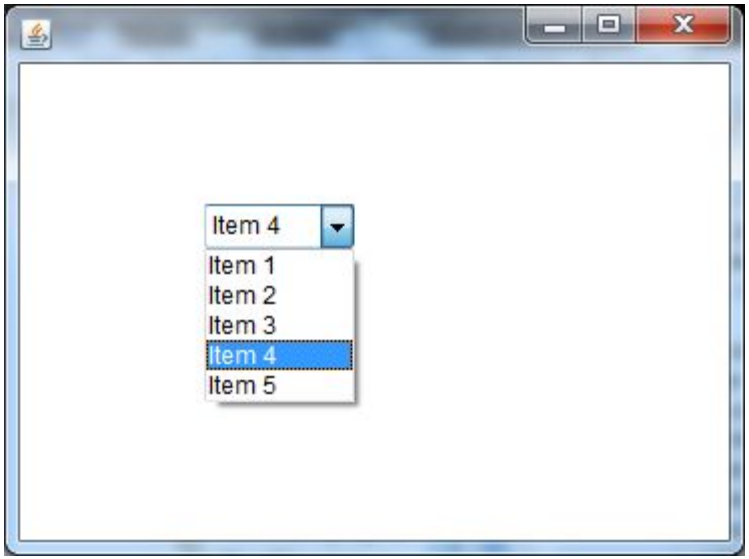


Choice lists/combo box

- The choice is a pull-down list. It is also called as combo box. The choice menu is used to display a list of choices for the user to select from.
- The choice menu is selected by mouse click on the choice control, a list of option drop down. This lets you select only one item at a time.
- The following constructor creates a choice list:
  - Choice();

EXAMPLE-

```
import java.awt.*;
public class ChoiceExample
{
      ChoiceExample(){
      Frame f= new Frame();
      Choice c=new Choice();
      c.setBounds(100,100, 75,75);
      c.add("Item 1");
      c.add("Item 2");
      c.add("Item 3");
      c.add("Item 4");
      c.add("Item 5");
      f.add(c);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
    }
public static void main(String args[])
{
  new ChoiceExample();
}
}
```



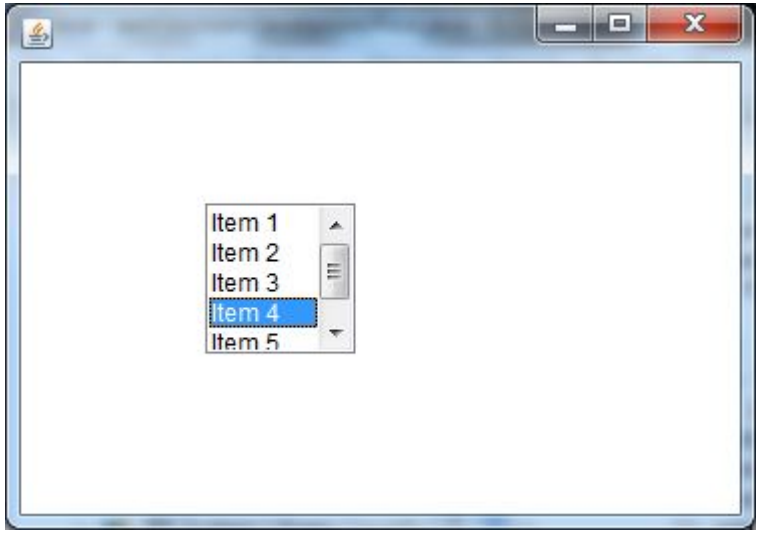
LISTS

- The List class provides a compact, multiple-choice, scrolling selection list. Unlike the Choice object, which shows only the single selected item in the menu, a List object can be constructed to show any number of choices in the visible window. It can also be created to allow multiple selections.
- List provides these constructors:
  - List( )
  - List(int *numRows*)
  - List(int *numRows*, boolean *multipleSelect*)
- The first version creates a List control that allows only one item to be selected at any one time.
- In the second form, the value of *numRows* specifies the number of entries in the list that will always be visible (others can be scrolled into view as needed).
- In the third form, if *multipleSelect* is true, then the user may select two or more items at a time. If it is false, then only one item may be selected.

Example-

- The object of List class represents a list of text items. By the help of list, user can choose either one item or multiple items. It inherits Component class.

```
import java.awt.*;
public class ListExample
{
    ListExample(){
        Frame f= new Frame();
        List l1=new List(5);
        l1.setBounds(100,100, 75,75);
        l1.add("Item 1");
        l1.add("Item 2");
        l1.add("Item 3");
        l1.add("Item 4");
        l1.add("Item 5");
        f.add(l1);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
public static void main(String args[])
{
  new ListExample();
}
}
```

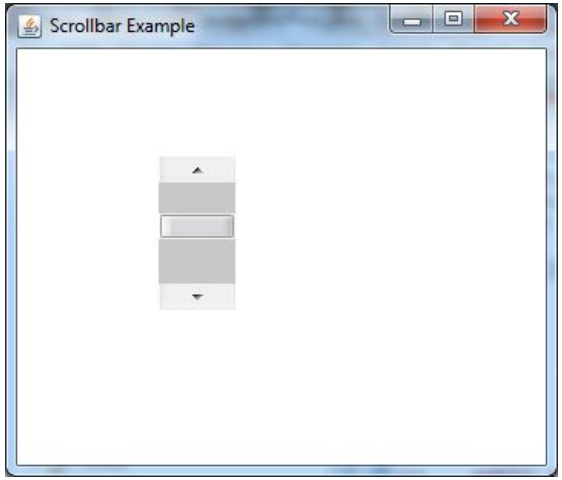


Scroll Bars

- Scroll bars are used to select continuous values between a specified minimum and maximum. Scroll bars may be oriented horizontally or vertically. Scrollbar defines the following constructors:
- Scrollbar( )
- Scrollbar(int style)
- Scrollbar(int style, int initialValue, int thumbSize, int min, int max)
- The first form creates a vertical scroll bar.
- The second and third forms allow you to specify the orientation of the scroll bar.
- If style is Scrollbar.VERTICAL, a vertical scroll bar is created. If style is Scrollbar.HORIZONTAL, the scroll bar is horizontal.
- In the third form of the constructor, the initial value of the scroll bar is passed in initialValue. It is the value where by default the scroll box is positioned.
- The number of units represented by the height of the thumb is passed in thumbSize.
- The minimum and maximum values for the scroll bar are specified by min and max.
- It is the range of values within which the scroll box moves. Beyond these values, the scroll box cannot move.

PROGRAM

```
import java.awt.*;
class ScrollbarExample{
ScrollbarExample(){
        Frame f= new Frame("Scrollbar Example");
        Scrollbar s=new Scrollbar();
        s.setBounds(100,100, 50,100);
        f.add(s);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
}
public static void main(String args[]){
      new ScrollbarExample();
}
```

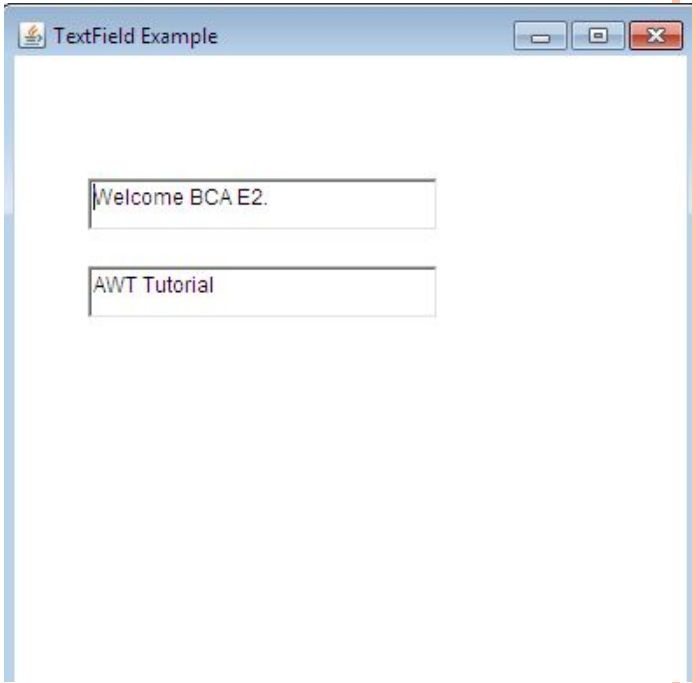


Text Field

- To accept the textual data from the user, AWT provides TextField. It is a subclass of the text components.
- TextField() – it construct an empty text field.
- TextField(String text)- it construct a text field with the initial content of text.
- TextField(int n)- construct empty text field with n number of columns.
- TextField(String text, int n)- construct a text field whose initial content is text with n number of columns.

Program

```
import java.awt.*;
class TextFieldExample{
public static void main(String args[]){
    Frame f= new Frame("TextField Example");
    TextField t1,t2;
    t1=new TextField("Welcome BCA E2.");
    t1.setBounds(50,100, 200,30);
    t2=new TextField("AWT Tutorial");
    t2.setBounds(50,150, 200,30);
    f.add(t1); f.add(t2);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```



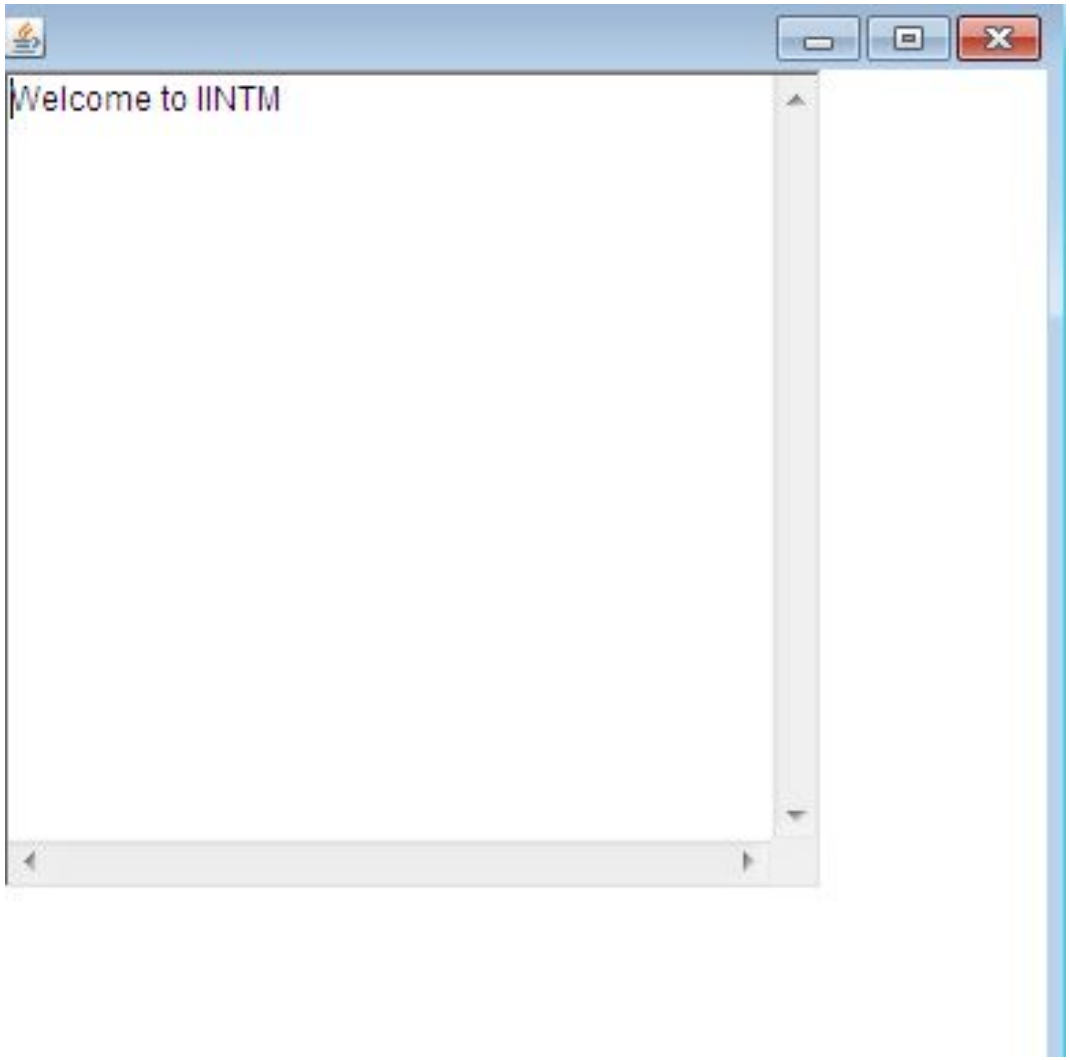
Text Area

- TextArea class handles the multiple lines of text.
- TextArea()- it construct an empty text area.
- TextArea(string text)- it construct a text area with the initial content of text.
- TextArea(int row, int col)- construct an empty text area with specified number of rows and columns.

Program-

```
import java.awt.*;
public class TextAreaExample
{
    TextAreaExample(){
      Frame f= new Frame();
         TextArea area=new TextArea("Welcome to IINTM");
      area.setBounds(10,30, 300,300);
      f.add(area);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
    }
public static void main(String args[])
{
  new TextAreaExample();
}
}
```

Output-



LayoutManagers

- 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 LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers.
- There are following classes that represents the layout managers:
  - java.awt.BorderLayout
  - java.awt.FlowLayout
  - java.awt.GridLayout
  - java.awt.CardLayout

Java FlowLayout

- The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.
- Fields of FlowLayout class:
  - **public static final int LEFT -** This value indicates that each row of components should be left-justified.
  - **public static final int RIGHT -** This value indicates that each row of components should be right-justified.
  - **public static final int CENTER -** This value indicates that each row of components should be centered.
  - **public static final 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.
  - **public static final 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.

CONTD..

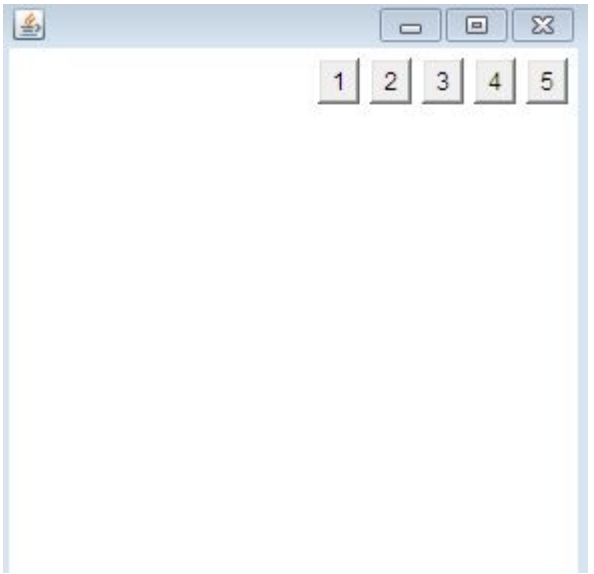
- Constructors of FlowLayout class:
  - **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
  - **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
  - **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap between the components..

Example of FlowLayout class

MyFlowLayout - Notepad

File Edit Format View Help

```
import java.awt.*;


public class MyFlowLayout{
Frame f;
MyFlowLayout(){
    f=new Frame();

    Button b1=new Button("1");
    Button b2=new Button("2");
    Button b3=new Button("3");
    Button b4=new Button("4");
    Button b5=new Button("5");

    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);

    f.setLayout(new FlowLayout(FlowLayout.RIGHT));
    //setting flow layout of right alignment

    f.setSize(300,300);
    f.setVisible(true);
}
public static void main(String[] args) {
    new MyFlowLayout();
}
}
```



Java GridLayout

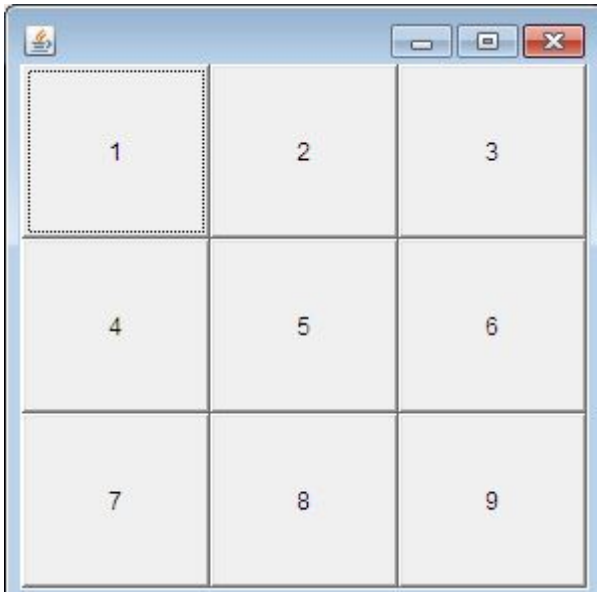
- The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.
- Constructors of GridLayout class
  - **GridLayout():** creates a grid layout with one column per component in a row.
  - **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
  - **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps between the components..

EXAMPLE

MyGridLayout - Notepad

File Edit Format View Help

```
import java.awt.*;

public class MyGridLayout{
Frame f;
MyGridLayout(){
    f=new Frame();

    Button b1=new Button("1");
    Button b2=new Button("2");
    Button b3=new Button("3");
    Button b4=new Button("4");
    Button b5=new Button("5");
    Button b6=new Button("6");
    Button b7=new Button("7");
    Button b8=new Button("8");
    Button b9=new Button("9");

    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
    f.add(b6);f.add(b7);f.add(b8);f.add(b9);

    f.setLayout(new GridLayout(3,3));
    //setting grid layout of 3 rows and 3 columns

    f.setSize(300,300);
    f.setVisible(true);
}
public static void main(String[] args) {
    new MyGridLayout();
}
}
```



BorderLayout

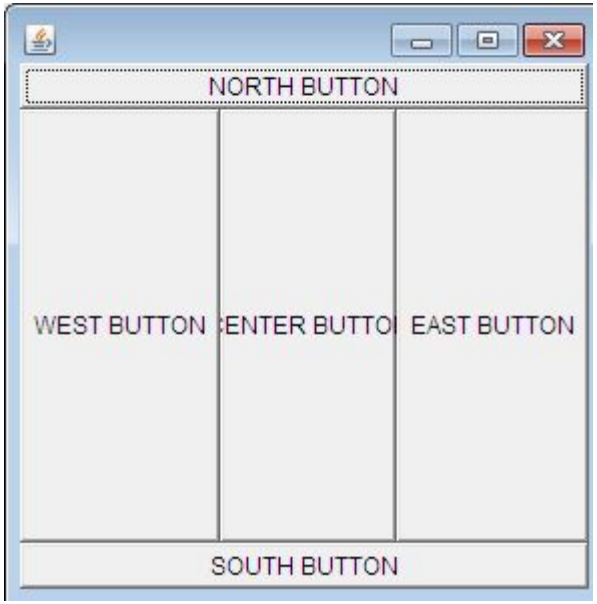
- The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:
  - **public static final int NORTH**
  - **public static final int SOUTH**
  - **public static final int EAST**
  - **public static final int WEST**
  - **public static final int CENTER**
- Constructors of BorderLayout class:
  - **BorderLayout():** creates a border layout but with no gaps between the components.
  - **BorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

Program

Border - Notepad

File Edit Format View Help

```
import java.awt.*;

public class Border{
Frame f;
Border(){
    f=new Frame();

    Button b1=new Button("NORTH BUTTON");;
    Button b2=new Button("SOUTH BUTTON");;
    Button b3=new Button("EAST BUTTON");;
    Button b4=new Button("WEST BUTTON");;
    Button b5=new Button("CENTER BUTTON");;

    f.add(b1,BorderLayout.NORTH);
    f.add(b2,BorderLayout.SOUTH);
    f.add(b3,BorderLayout.EAST);
    f.add(b4,BorderLayout.WEST);
    f.add(b5,BorderLayout.CENTER);

    f.setSize(300,300);
    f.setVisible(true);
}
public static void main(String[] args) {
    new Border();
}
}
```



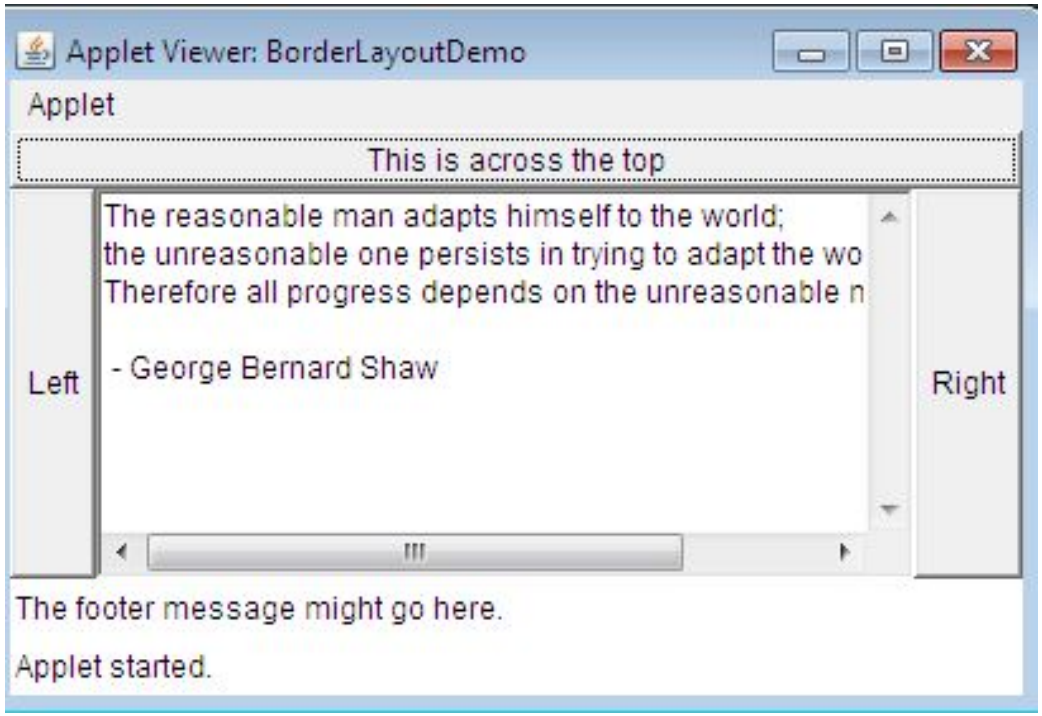
Using Applets

BorderLayoutDemo - Notepad

File Edit Format View Help

```
import java.awt.*;
import java.applet.*;
/*
<applet code="BorderLayoutDemo" width=400 height=200>
</applet>
*/
public class BorderLayoutDemo extends Applet {

public void init() {
BorderLayout bl=new BorderLayout();
setLayout(bl);
Button b=new Button("This is across the top");
add(b, BorderLayout.NORTH);
Label l=new Label("The footer message might go here.");
add(l, BorderLayout.SOUTH);
Button b1=new Button("Right");
add(b1, BorderLayout.EAST);
Button b2=new Button("Left");
add(b2, BorderLayout.WEST);

String msg = "The reasonable man adapts " +
"himself to the world;\n" +
"the unreasonable one persists in " +
"trying to adapt the world to himself.\n" +"Therefore all progress depends " +
"on the unreasonable man.\n\n" +
" - George Bernard Shaw\n\n";

TextArea ta= new TextArea(msg);
add(ta, BorderLayout.CENTER);
}
}
```

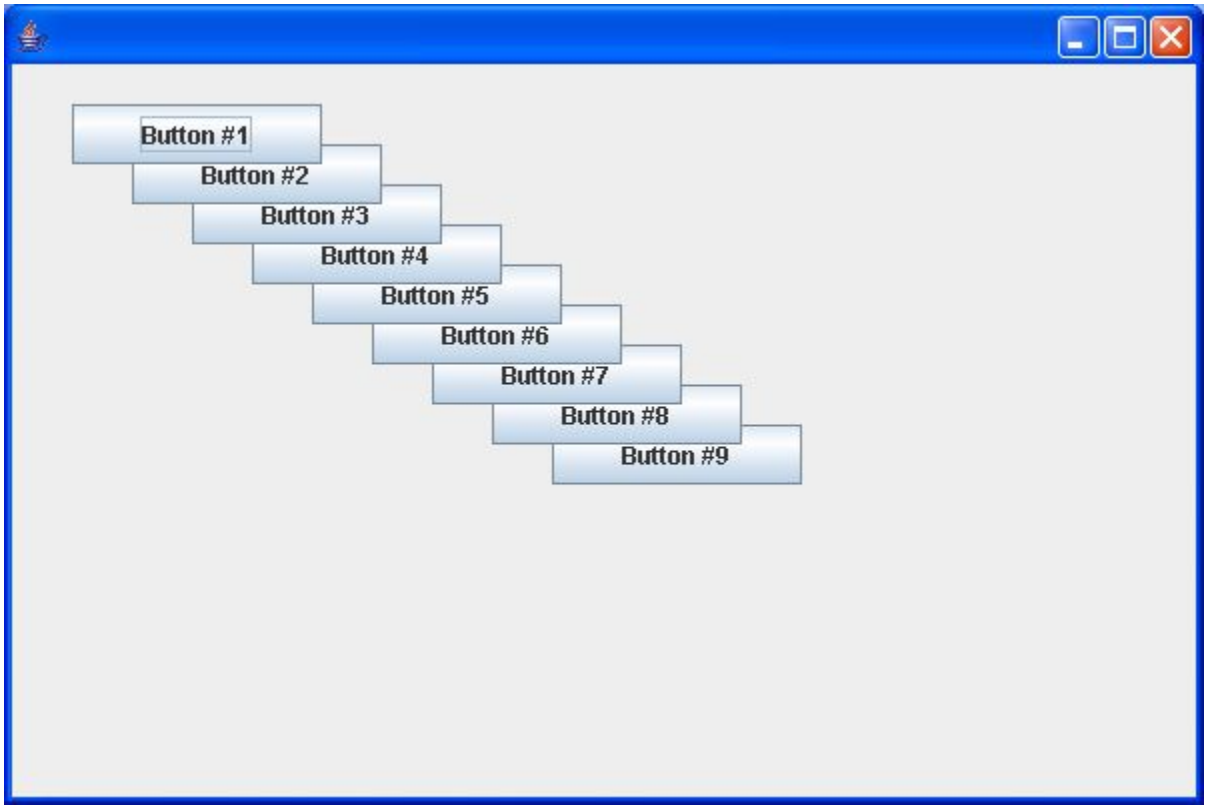
OUTPUT


Applet Viewer: BorderLayoutDemo
Applet
This is across the top
The reasonable man adapts himself to the world;
the unreasonable one persists in trying to adapt the wo
Therefore all progress depends on the unreasonable n
- George Bernard Shaw
Left
Right
The footer message might go here.
Applet started.

Card Layout

- The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.
- Constructors of CardLayout class
  - **CardLayout():** creates a card layout with zero horizontal and vertical gap.
  - **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap between the components.
- Commonly used methods of CardLayout class
  - **public void next(Container parent):** is used to flip to the next card of the given container.
  - **public void previous(Container parent):** is used to flip to the previous card of the given container.
  - **public void first(Container parent):** is used to flip to the first card of the given container.
  - **public void last(Container parent):** is used to flip to the last card of the given container.
  - **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

Diagram-


Button #1
Button #2
Button #3
Button #4
Button #5
Button #6
Button #7
Button #8
Button #9

Event Handling

- **What is an Event?**
- Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as 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 list, scrolling the page are the activities that causes an event to happen.

- **Types of Event:**
  - **Foreground Events**
  - **Background Events**

Contd…

- **Foreground Events** - Those events which require the direct interaction of 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** - Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

What is Event Handling?

- Event Handling is the mechanism that controls the event and decides what should happen if an event occurs.
- This mechanism have 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.

CONTD…

- The Delegation Event Model has the following key participants namely:
- **Events -** An event is a change in state of an object.
- **Source** - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide as with classes for source object.
- **Listener** - It is also known as event handler. Listener is responsible for generating response to an 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 process the event an then returns.

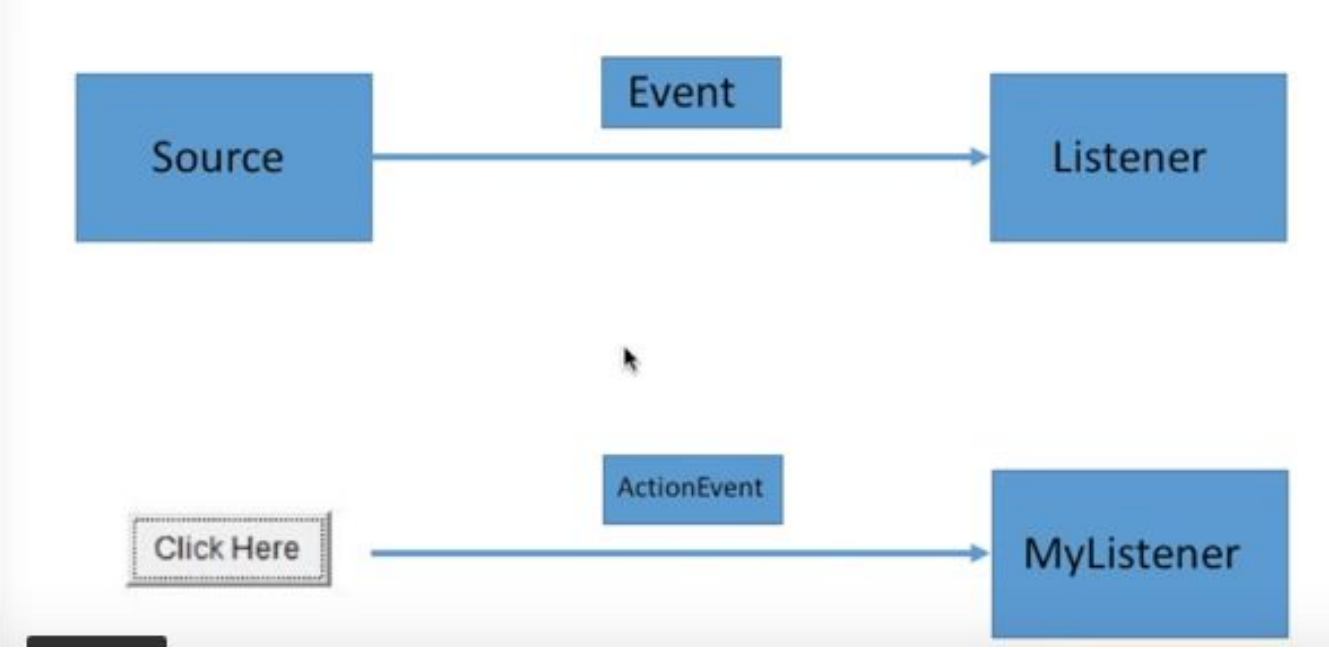
Advantage

- 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.

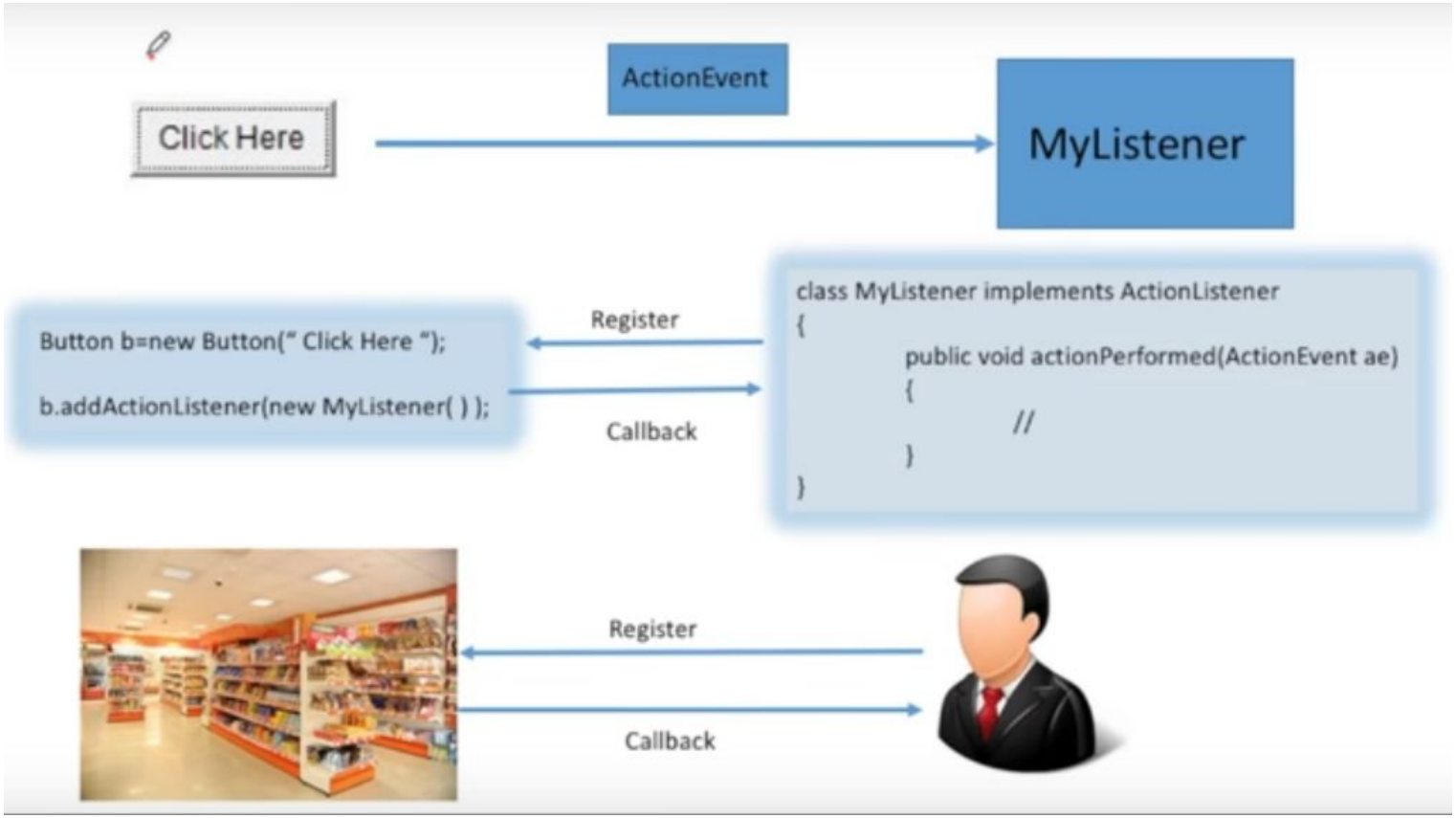
How events are handled?

- A source generates an Event and send it to one or more listeners registered with the source.
- Once event is received by the listener, they process the event and then return.
- Events are supported by a number of Java packages, like **java.util**, **java.awt** and **java.awt.event**.

Example-



CONTD..


Click Here
ActionEvent
MyListener
Button b=new Button(" Click Here ");
b.addActionListener(new MyListener( ));
Register
Callback
class MyListener implements ActionListener
{
public void actionPerformed(ActionEvent ae)
{
//
}
}
Register
Callback

Steps involved in event handling

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

Types of listeners

- **public void add*Type*Listener(*Type*Listener *el*)**
- Here, *Type* is the name of the event and *el* is a reference to the event listener. For example, the method that registers a keyboard event listener is called addKeyListener( ). The method that registers a mouse motion listener is called addMouseMotionListener( ).
- A source must also provide a method that allows a listener to unregister an interest in a specific type of event. The general form of such a method is this:
- **public void remove*Type*Listener(*Type*Listener *el*)**
- Here, *Type* is the name of the event and *el* is a reference to the event listener. For example, to remove a keyboard listener, you would call **removeKeyListener( )**.

CONTD..

- **Event Listeners-**
- A *listener* is an object that is notified when an event occurs. It has two major requirements. First, it must have been registered with one or more sources to receive notifications about specific types of events. Second, it must implement methods to receive and process these notifications.
- The methods that receive and process events are defined in a set of interfaces found in java.awt.event.
- **Event Classes-**
- The classes that represent events are called Event Classes. At the root of the Java event class hierarchy is **EventObject**, which is in **java.util**. It is the superclass for all events. Its one constructor is shown here:
  - **EventObject(Object *src*)**
- Here, *src* is the object that generates this event.

Contd…

- **EventObject** contains two methods:
  - **getSource( )** and **toString( )**.
  - The **getSource( )** method returns the source of the event. Its general form is shown here:
    - **Object getSource( )**
  - As expected, **toString( )** returns the string equivalent of the event.

Various Event classes

Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract super class for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.

Table 20-1. *Main Event Classes in* java.awt.event

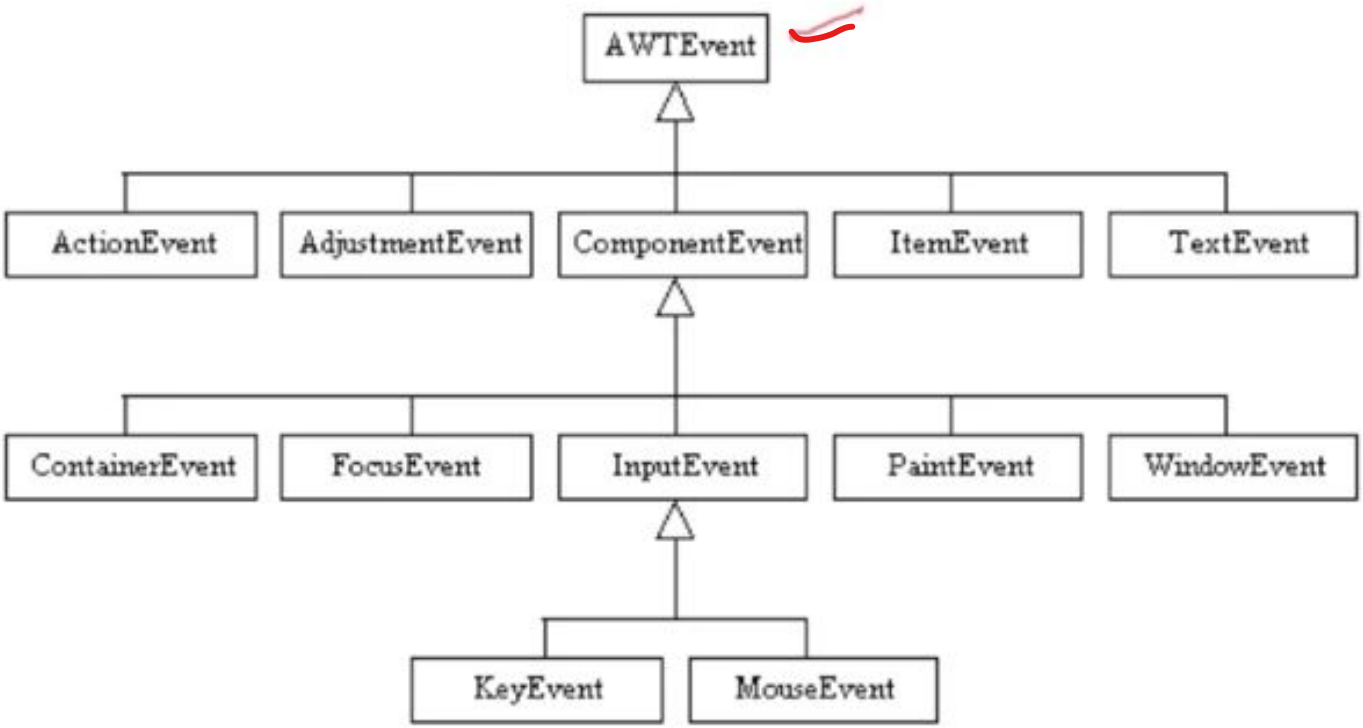
CONTD…

Event Class	Description
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when the mouse wheel is moved. (Added by Java 2, version 1.4)
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Table 20-1. *Main Event Classes in* java.awt.event (continued)

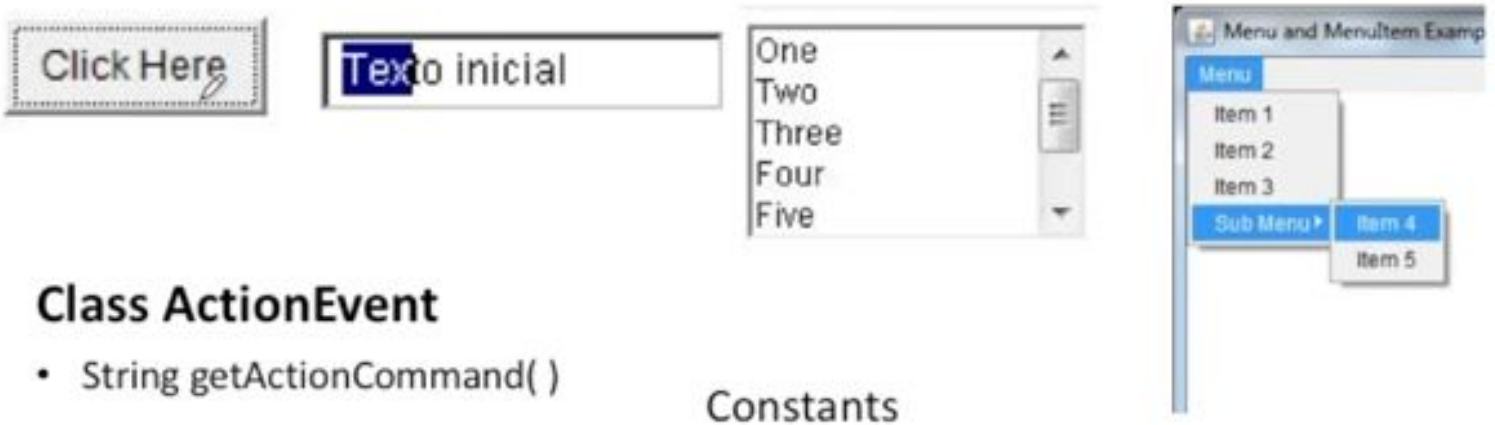
Note- Iconifying means substituting a small icon on the desktop for the window *deiconifying* means the opposite.

Event class hierarchy-



Action event class

ActionEvent



Class ActionEvent

- String getActionCommand( )
- int getModifiers( )
- String paramString( )

Constants

- **ActionEvent.ALT_MASK** -- The alt modifier.
- **ActionEvent.CTRL_MASK** -- The control modifier.
- **ActionEvent.META_MASK** -- The meta modifier.
- **ActionEvent.SHIFT_MASK** -- The shift modifier.

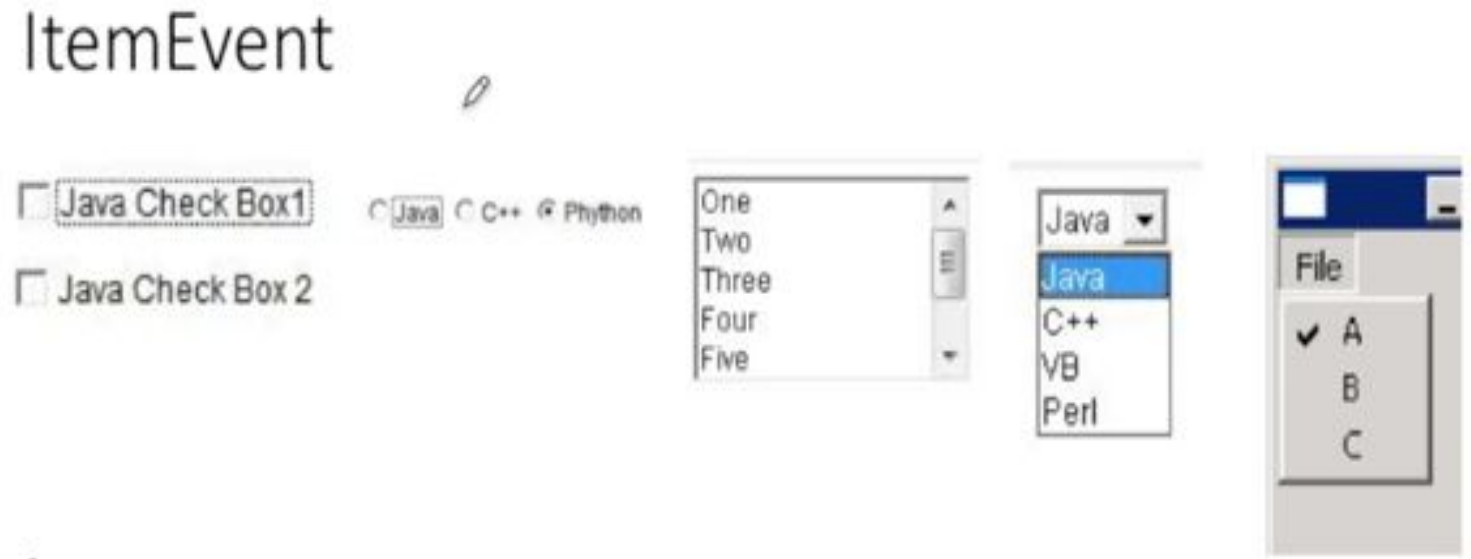
ActionEvent Class-

- An **ActionEvent** is generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
- **ActionEvent** has these three constructors:
  - ActionEvent(Object *src*, int *type*, String *cmd*)
  - ActionEvent(Object *src*, int *type*, String *cmd*, int *modifiers*)
  - ActionEvent(Object *src*, int *type,* String *cmd,* long *when*, int *modifiers*)
- Here, *src* is a reference to the object that generated this event. The type of the event is specified by *type*, and its command string is *cmd*.
- The argument *modifiers* indicates which modifier keys (ALT, CTRL, META, and/or SHIFT) were pressed when the event was generated.
- The *when* parameter specifies when the event occurred.

Event Class methods

- **getActionCommand() –** Returns the command string associated with this action.
- For example, when a button is pressed, an action event is generated that has a command name equal to the label on that button.
- **getModifiers()** - Returns the modifier keys held down during this action event.
- **getWhen()**- Returns the timestamp of when this event occurred.
- **paramString()**- Returns a parameter string identifying this action event.

ITEM EVENT CLASS



Class ItemEvent

- Object getItem( )
- int getStateChange()
- String paramString()

Constants

ItemEvent.SELECTED

ItemEvent.DESELECTED

ItemEvent Class

- An ItemEvent is generated when a check box or a list item is clicked or when a checkable menu item is selected or deselected. There are two types of item events:
  - DESELECTED The user deselected an item.
  - SELECTED The user selected an item.
- **ItemEvent** has this constructor:
- ItemEvent(ItemSelectable *src*, int *type*, Object *entry*, int *state*)
- Here, *src* is a reference to the component that generated this event.
- For example, this might be a list or choice element. The type of the event is specified by *type*. The specific item that generated the item event is passed in *entry*. The current state of that item is in *state*.
- The **getItem( )** method can be used to obtain a reference to the item that generated an event. Its signature is shown here:
  - Object getItem( )

KeyEvent Class

- A KeyEvent is generated when keyboard input occurs. There are three types of key events, which are identified by these integer constants:
  - KEY_PRESSED,
  - KEY_RELEASED
  - KEY_TYPED
- The first two events are generated when any key is pressed or released. The last event occurs only when a character is generated.
- Not all key presses result in characters.
- For example, pressing the SHIFT key does not generate a character.
- The **KeyEvent** class defines several methods, but the most commonly used ones are **getKeyChar( )**, which returns the character that was entered, and **getKeyCode( )**, which returns the key code. Their general forms are shown here:
  - char getKeyChar( )
  - int getKeyCode( )

Mouse Event Class

- There are eight types of mouse events. The MouseEvent class defines the following integer constants that can be used to identify them:
  - MOUSE_CLICKED The user clicked the mouse.
  - MOUSE_DRAGGED The user dragged the mouse.
  - MOUSE_ENTERED The mouse entered a component.
  - MOUSE_EXITED The mouse exited from a component.
  - MOUSE_MOVED The mouse moved.
  - MOUSE_PRESSED The mouse was pressed.
  - MOUSE_RELEASED The mouse was released.
  - MOUSE_WHEEL The mouse wheel was moved
- **MouseEvent** is a subclass of **InputEvent**. Here is one of its constructors.

CONTD…

- MouseEvent(Component *src*, int *type*, long *when*, int *modifiers*, int *x*, int *y*, int *clicks*, boolean *triggersPopup*)
- Here, *src* is a reference to the component that generated the event.
- The type of the event is specified by *type*.
- The system time at which the mouse event occurred is passed in *when*.
- The *modifiers* argument indicates which modifiers were pressed when a mouse event occurred.
- The coordinates of the mouse are passed in *x* and *y*. The click count is passed in *clicks*.
- The *triggersPopup* flag indicates if this event causes a pop-up menu to appear on this platform.
- The most commonly used methods in this class are **getX( )** and **getY( )**. These return the X and Y coordinates of the mouse when the event occurred. Their forms are shown here:
  - int getX( )
  - int getY( )

Sources of Events

Event Source	Description
Button	Generates action events when the button is pressed.
Checkbox	Generates item events when the check box is selected or deselected.
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
Menu Item	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Scrollbar	Generates adjustment events when the scroll bar is manipulated.
Text components	Generates text events when the user enters a character.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Table 20-2. *Event Source Examples*

Event Listener Interfaces

- The Event listener represent the interfaces responsible to handle events. Java provides us various Event listener classes.
- Every method of an event listener method has a single argument as an object which is subclass of EventObject class.
- For example, mouse event listener methods will accept instance of MouseEvent, where MouseEvent derives from EventObject.
- **EventListner interface-**
- It is a marker interface which every listener interface has to extend.This is defined in java.util package.

public interface EventListener

Following is the list of commonly used event listeners.

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.

Contd…

8	**AdjustmentListener** This interface is used for receiving the adjusmtent 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.

ActionListener Interface

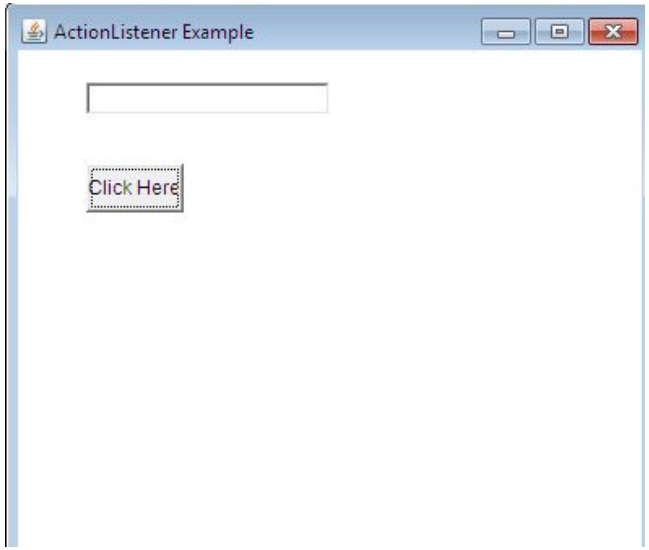
- The Java ActionListener is notified whenever you click on the button or menu item.
- It is notified against **ActionEvent** class.
- The class which processes the ActionEvent should implement this interface.
- The object of that class must be registered with a component.
- The object can be registered using the **addActionListener()** method.
- Interface methods-
  - **void actionPerformed(ActionEvent e)**
  - It is invoked when an action occurs.
  - It is invoked automatically whenever you click on the registered component.

Java ActionListener Example: On Button click

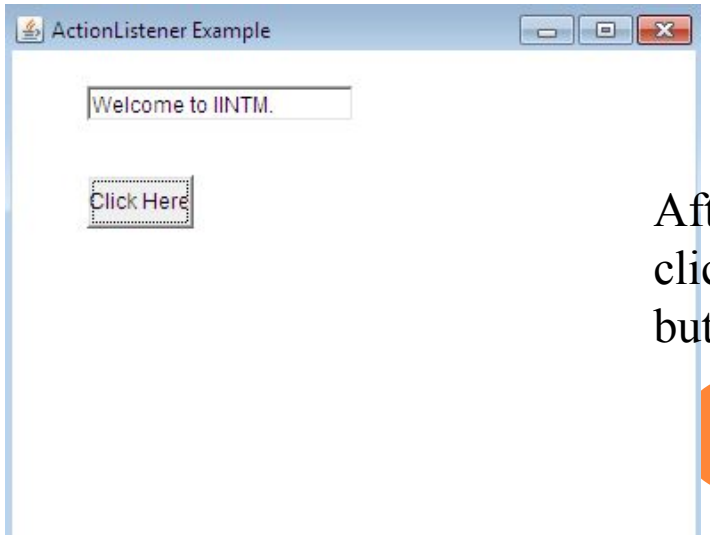
ActionListenerExample - Notepad

File Edit Format View Help

```
import java.awt.*;
import java.awt.event.*;
public class ActionListenerExample {
public static void main(String[] args) {
    Frame f=new Frame("ActionListener Example");
    final TextField tf=new TextField();
    tf.setBounds(50,50, 150,20);
    Button b=new Button("Click Here");
    b.setBounds(50,100,60,30);

    b.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
            tf.setText("Welcome to IINTM.");
    }
    });
    f.add(b);f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```



Before clicking button



After clicking button

ItemListener Interface

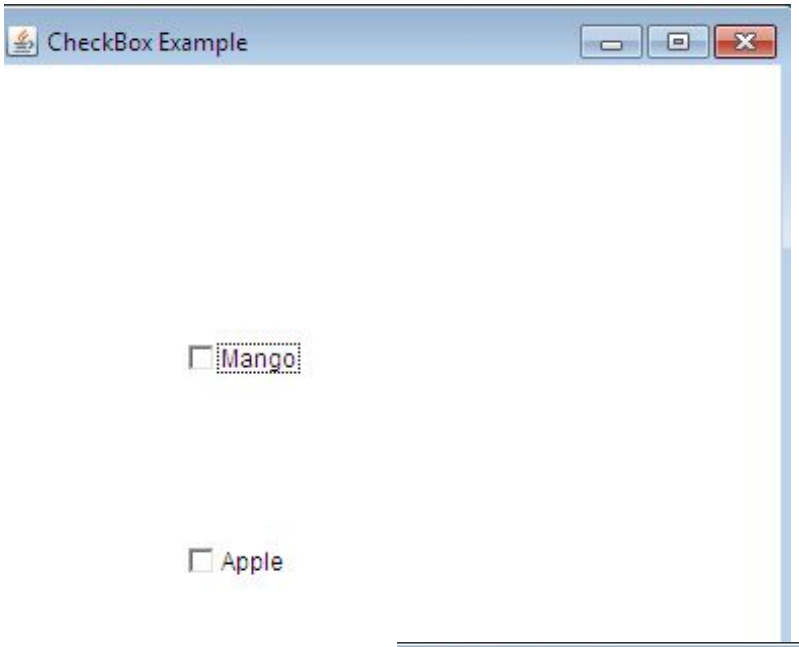
- The Java ItemListener is notified whenever you click on the checkbox.
- It is notified against **ItemEvent** class.
- The class which processes the ItemEvent should implement this interface.
- The object of that class must be registered with a component.
- The object can be registered using the **addItemListener()** method.
- Interface methods-
  - **void itemStateChanged(ItemEvent e)**
  - It is invoked when an item has been selected or deselected by the user.
  - This method is invoked automatically whenever you click or unclick on the registered checkbox component.

Java ItemListener Example

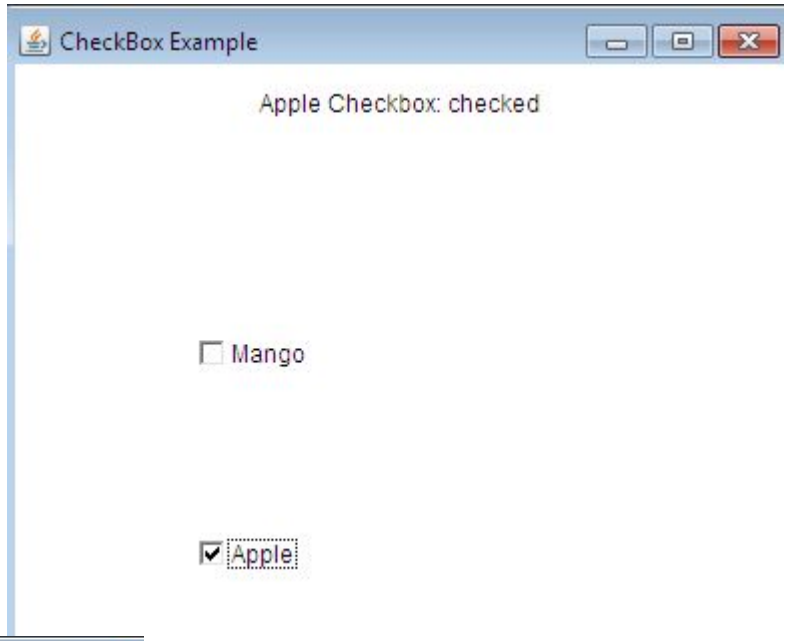
```
import java.awt.*;
import java.awt.event.*;
public class ItemListenerExample implements ItemListener{
    Checkbox checkBox1,checkBox2;
    Label label;
    ItemListenerExample(){
        Frame f= new Frame("CheckBox Example");
        label = new Label();
        label.setAlignment(Label.CENTER);
        label.setSize(400,100);
        checkBox1 = new Checkbox("Mango");
        checkBox1.setBounds(100,100, 50,50);
        checkBox2 = new Checkbox("Apple");
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1); f.add(checkBox2); f.add(label);
        checkBox1.addItemListener(this);
        checkBox2.addItemListener(this);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
     }
    public void itemStateChanged(ItemEvent e) {
        if(e.getSource()==checkBox1)
            label.setText("Mango Checkbox: "
            + (e.getStateChange()==1?"checked":"unchecked"));
        if(e.getSource()==checkBox2)
        label.setText("Apple Checkbox: "
        + (e.getStateChange()==1?"checked":"unchecked"));
     }
public static void main(String args[])
{
    new ItemListenerExample();
}
}
```

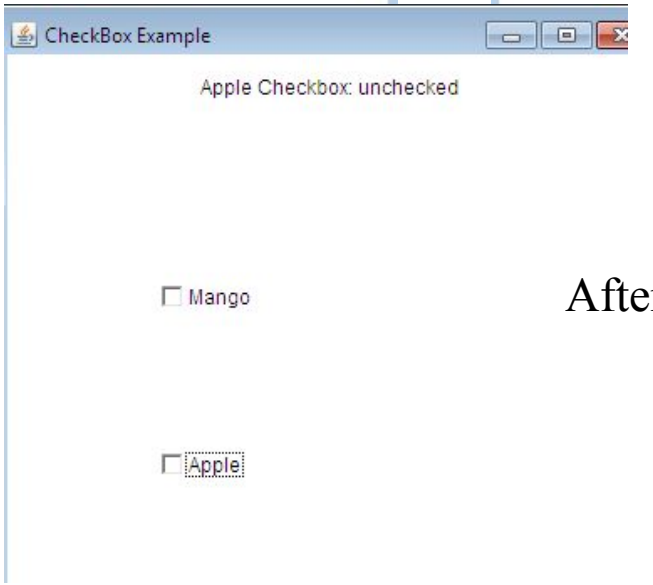
OUTPUT-

Before clicking



After selecting apple





After deselecting apple

KeyListener Interface

- The Java KeyListener is notified whenever you change the state of key.
- It is notified against **KeyEvent** class.
- 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 methods-
  - **void keyPressed(KeyEvent e)**
  - It is invoked when a key has been pressed.
  - **void keyReleased(KeyEvent e)**
  - It is invoked when a key has been released.
  - **void keyTyped(KeyEvent e)**
  - It is invoked when a key has been typed.
- **Note- keyPressed** is fired whenever any key press occurs whereas **keyTyped** is fired when a key is pressed that can be converted into a unicode character.
- For example keys like shift, capslock, down arrow, up arrow etc.. do not have any Unicode so for them, keyPressed occurs.

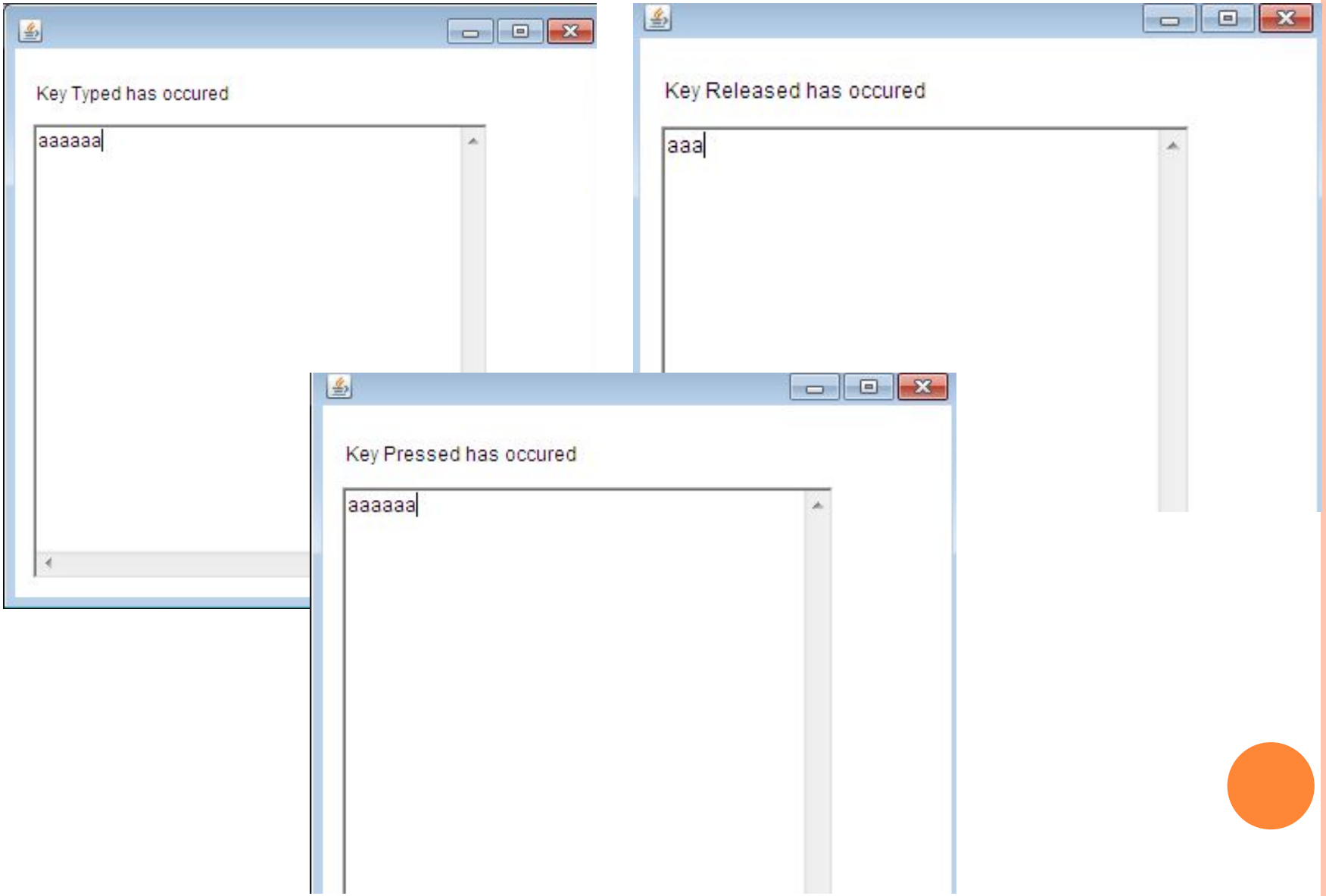
Java KeyListener Example

KeyListenerExample - Notepad

File Edit Format View Help

```
import java.awt.*;
import java.awt.event.*;
public class KeyListenerExample extends Frame implements KeyListener{
    Label l;
    TextArea area;
    KeyListenerExample(){

        l=new Label();
        l.setBounds(20,50,800,20);
        area=new TextArea();
        area.setBounds(20,80,300, 300);
        area.addKeyListener(this);

        add(l);add(area);
        setSize(400,400);
        setLayout(null);
        setVisible(true);
    }
    public void keyPressed(KeyEvent e) {
        l.setText("Key Pressed has occured");
    }
    public void keyReleased(KeyEvent e) {
        l.setText("Key Released has occured");
    }
    public void keyTyped(KeyEvent e) {
        l.setText("Key Typed has occured");
    }

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

OUTPUT-


Key Typed has occured
aaaaaa
Key Released has occured
aaa
Key Pressed has occured
aaaaaa

MouseListener Interface

- The Java MouseListener is notified whenever you change the state of mouse.
- It is notified against **MouseEvent** class.
- 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.
- Methods of MouseListener interface
  - **void mouseClicked(MouseEvent e)**
  - It is invoked when the mouse button has been clicked (pressed and released) on a component.
  - **void mouseEntered(MouseEvent e)**
  - It is invoked when the mouse enters a component.

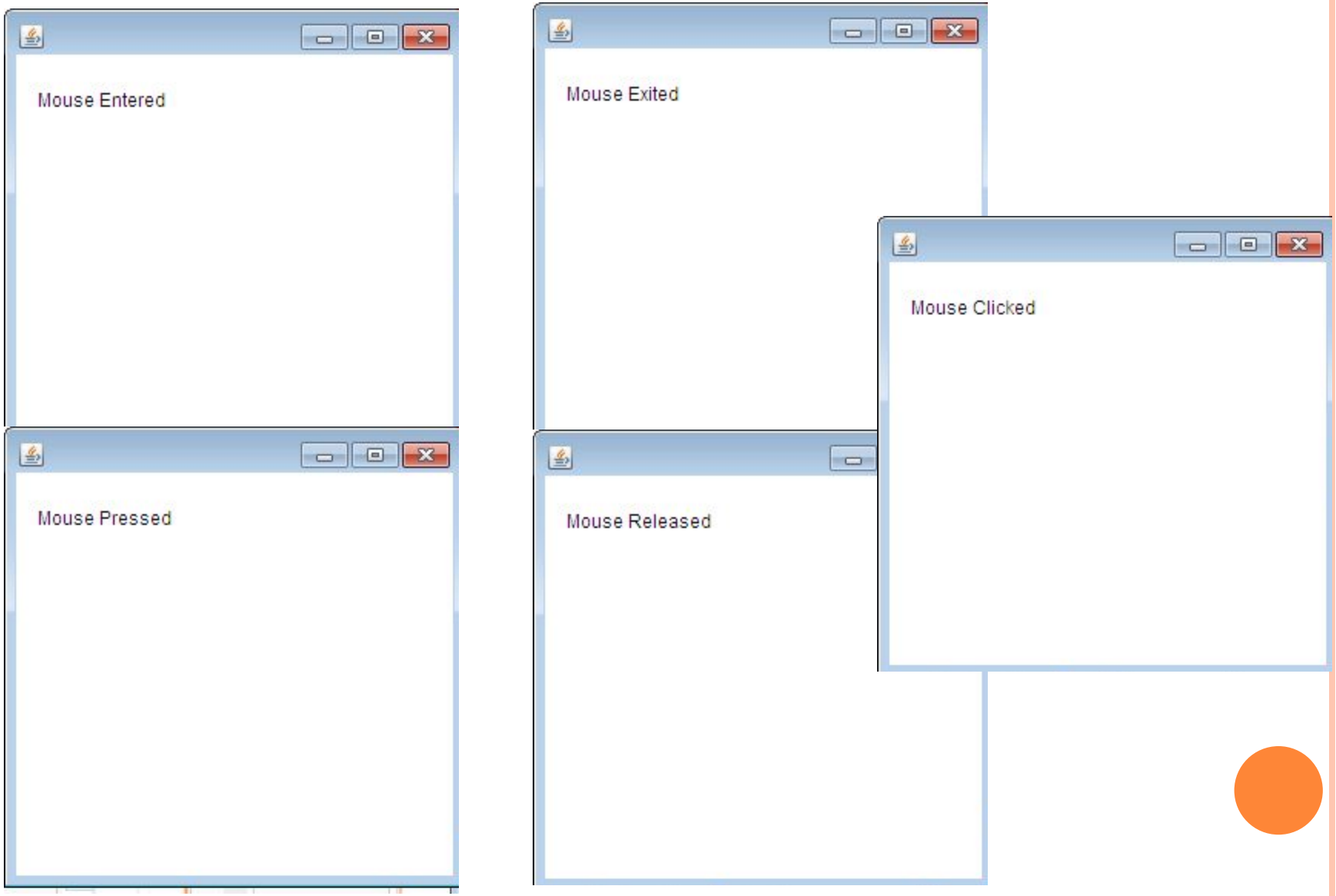
Contd…

- **void mouseExited(MouseEvent e)**
- It is invoked when the mouse exits a component.
- **void mousePressed(MouseEvent e)**
- It is invoked when a mouse button has been pressed on a component.
- **void mouseReleased(MouseEvent e)**
- It is invoked when a mouse button has been released on a component.

MouseListener Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample extends Frame implements MouseListener{
    Label l;
    MouseListenerExample(){
        addMouseListener(this);

        l=new Label();
        l.setBounds(20,50,100,20);
        add(l);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public void mouseClicked(MouseEvent e) {
        l.setText("Mouse Clicked");
    }
    public void mouseEntered(MouseEvent e) {
        l.setText("Mouse Entered");
    }
    public void mouseExited(MouseEvent e) {
        l.setText("Mouse Exited");
    }
    public void mousePressed(MouseEvent e) {
        l.setText("Mouse Pressed");
    }
    public void mouseReleased(MouseEvent e) {
        l.setText("Mouse Released");
    }
public static void main(String[] args) {
    new MouseListenerExample();
}
}
```

Output-


Mouse Entered
Mouse Exited
Mouse Clicked
Mouse Pressed
Mouse Released

DIFFERENCE BETWEEN MOUSE CLICKED AND MOUSE PRESSED

- mouseClicked is when the mouse button has been pressed and released.
- mousePressed is when the mouse button has been pressed. You dont need to realese for it to work.

MouseMotionListener Interface

- The Java MouseMotionListener is notified whenever you move or drag mouse.
- It is notified against **MouseEvent** class.
- The interface **MouseMotionListener** is used for receiving mouse motion events on a component.
- The class that process mouse motion events needs to implements this interface.
- Interface methods-
  - **void mouseDragged(MouseEvent e)**
  - It is invoked when a mouse button is pressed on a component and then dragged.
  - **void mouseMoved(MouseEvent e)**
  - It is invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.

Java MouseMotionListener Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseMotionListenerExample extends Frame implements MouseMotionListener{
    MouseMotionListenerExample(){
        addMouseMotionListener(this);

        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
public void mouseDragged(MouseEvent e) {
    Graphics g=getGraphics();
    g.setColor(Color.BLUE);
    g.fillOval(e.getX(),e.getY(),20,20);
}
public void mouseMoved(MouseEvent e) {}

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

Output-

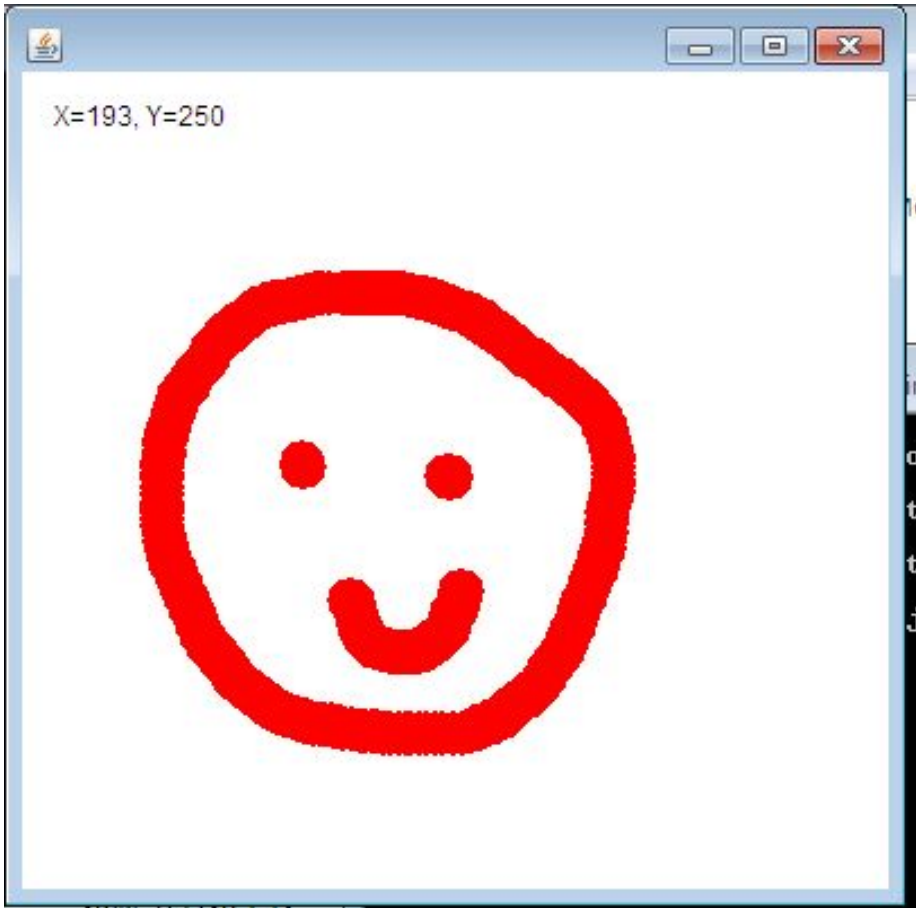
Wherever mouse was dragged blue color can be seen there.

Java MouseMotionListener Example 2

Paint - Notepad

File Edit Format View Help

```
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
public class Paint extends Frame implements MouseMotionListener{
    Label l;
    Color c=Color.BLUE;
    Paint(){
    l=new Label();
    l.setBounds(20,40,100,20);
    add(l);

    addMouseMotionListener(this);

    setSize(400,400);
    setLayout(null);
    setVisible(true);
}
public void mouseDragged(MouseEvent e) {
    l.setText("X="+e.getX()+", Y="+e.getY());
    Graphics g=getGraphics();
    g.setColor(Color.RED);
    g.fillOval(e.getX(),e.getY(),20,20);
}
public void mouseMoved(MouseEvent e) {
    l.setText("X="+e.getX()+", Y="+e.getY());
}
public static void main(String[] args) {
    new Paint();
}
}
```



In this example, when mouse was dragged red color appears & on the movement coordinates of X axis & Y axis are captured & printed as a label.

WindowListener Interface

- The Java WindowListener is notified whenever you change the state of window.
- It is notified against **WindowEvent** class.
- 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 Methods-
  - **void windowActivated(WindowEvent e)**
  - Invoked when the Window is set to be the active Window.
  - **void windowClosed(WindowEvent e)**
  - Invoked when a window has been closed as a result of calling dispose on the Window.

Contd…

- **void windowClosing(WindowEvent e)**
- Invoked when the user attempts to close the Window from the Window's system menu.
- **void windowDeactivated(WindowEvent e)**
- Invoked when a Window is no longer the active Window.
- **void windowDeiconified(WindowEvent e)**
- Invoked when a Window is changed from a minimized to a normal state.
- **void windowIconified(WindowEvent e)**
- Invoked when a Window is changed from a normal to a minimized state.
- **void windowOpened(WindowEvent e)**
- Invoked the first time a Window is made visible.

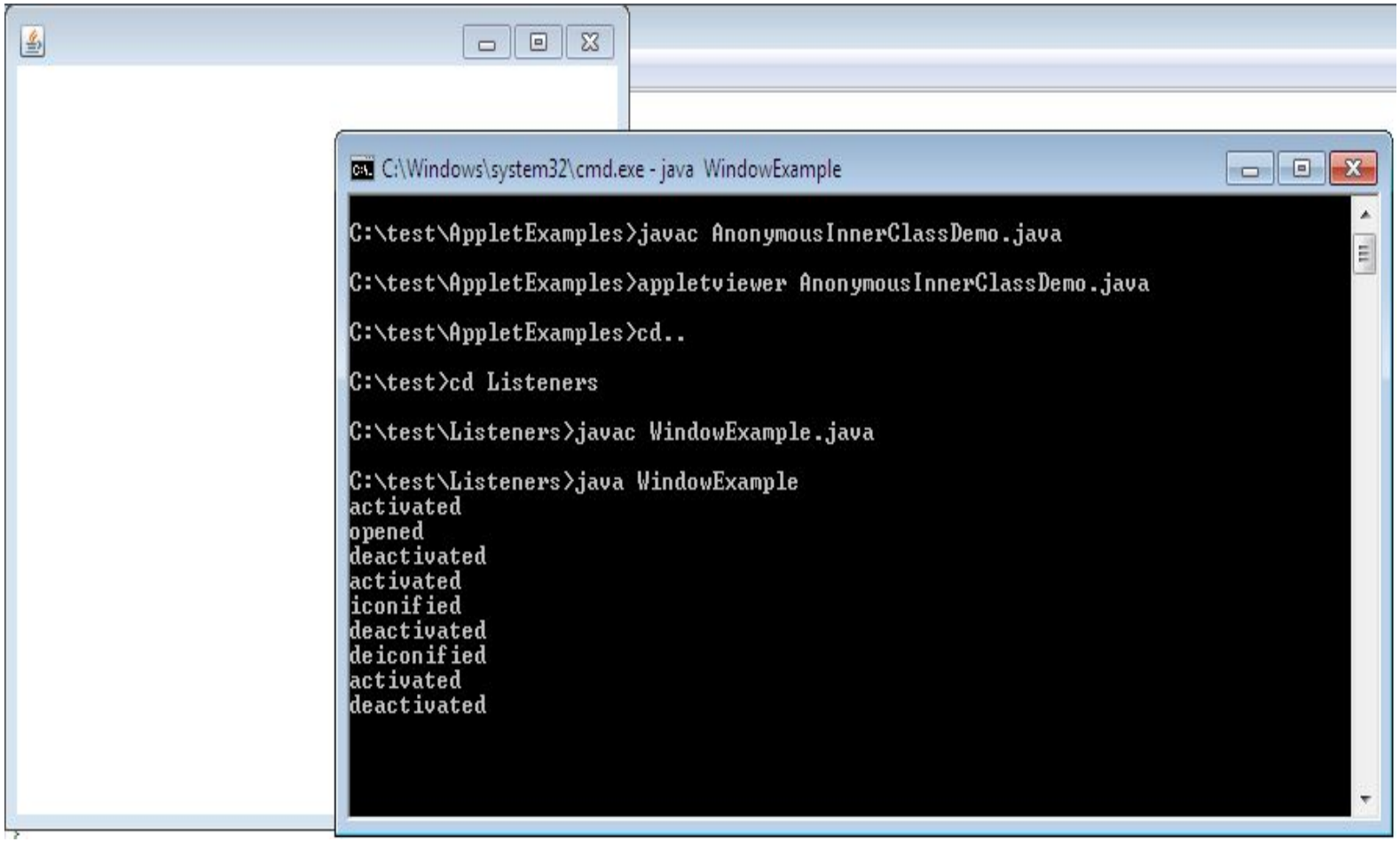
Java WindowListener Example

WindowExample - Notepad

File Edit Format View Help

```
import java.awt.*;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
public class WindowExample extends Frame implements WindowListener{
    WindowExample(){
        addWindowListener(this);

        setSize(400,400);
        setLayout(null);
        setVisible(true);
    }

public static void main(String[] args) {
    new WindowExample();
}
public void windowActivated(WindowEvent arg0) {
    System.out.println("activated");
}
public void windowClosed(WindowEvent arg0) {
    System.out.println("closed");
}
public void windowClosing(WindowEvent arg0) {
    System.out.println("closing");
    dispose();
}
public void windowDeactivated(WindowEvent arg0) {
    System.out.println("deactivated");
}
public void windowDeiconified(WindowEvent arg0) {
    System.out.println("deiconified");
}
public void windowIconified(WindowEvent arg0) {
    System.out.println("iconified");
}
public void windowOpened(WindowEvent arg0) {
    System.out.println("opened");
}
}
```

Output-


C:\Windows\system32\cmd.exe - java WindowExample
C:\test\AppletExamples>javac AnonymousInnerClassDemo.java
C:\test\AppletExamples>appletviewer AnonymousInnerClassDemo.java
C:\test\AppletExamples>cd..
C:\test>cd Listeners
C:\test\Listeners>javac WindowExample.java
C:\test\Listeners>java WindowExample
activated
opened
deactivated
activated
iconified
deactivated
deiconified
activated
deactivated

Event handling using applets example

Handling buttons events

Action - Notepad

File Edit Format View Help

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

/* <applet code="A.class" width=200 height=200>
</applet> */

public class A extends Applet implements ActionListener
{
 Button b;
 public void init()
  {
    b=new Button("Click Me");
    add(b);
    b.addActionListener(this);
  }
 public void actionPerformed(ActionEvent e)
  {
        if(e.getSource()==b)
        {
        setBackgound(Color.blue);
                }
        }
   }
```

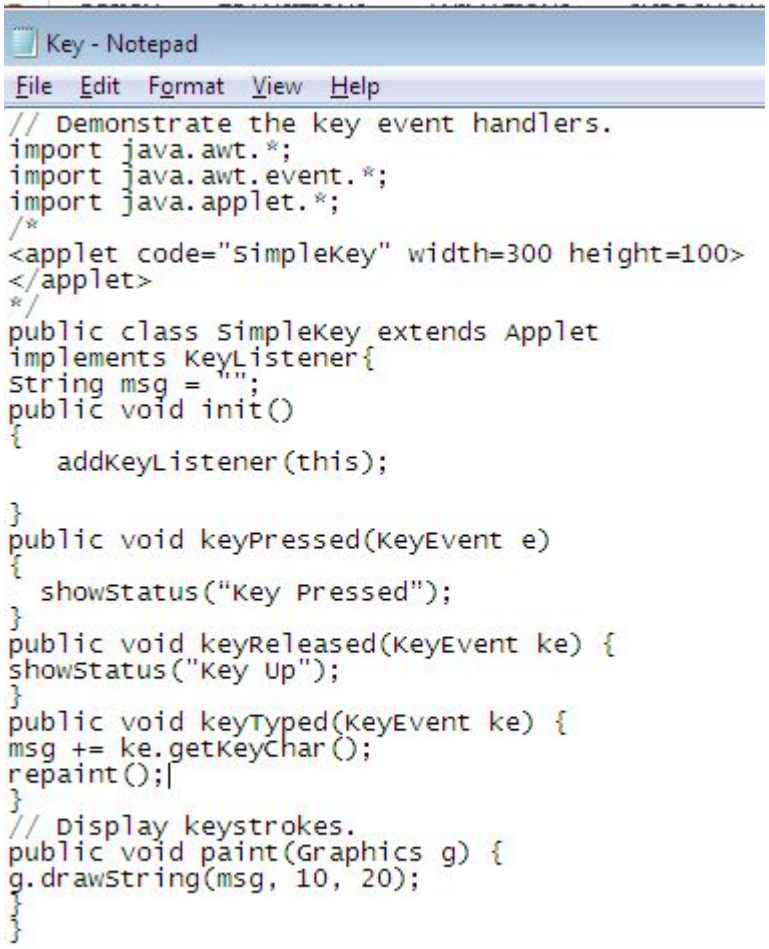
Handling Mouse Event

Mouse - Notepad

File Edit Format View Help

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="MouseEvents" width=300 height=100></applet>
*/
public class MouseEvents extends Applet implements MouseListener,MouseMotionListener
{
   String msg;
   int mousex=0, mousey=0;
  public void init()
        {
            addMouseListener(this);
        }

  public void mouseClicked(MouseEvent e)
        {
            mousex=e.getX();
            mousey=e.getY();
           msg="mouse clicked";
        }
public void mouseMoved(MouseEvent e)
{
   showStatus("mouse moving at + e.getX() + ", " + e.getY());
}
public void mouseEntered(MouseEvent me)
{
}
void mouseExited(MouseEvent me)
{
}
void mousePressed(MouseEvent me)
{
}
void mouseReleased(MouseEvent me)
{
}
void mouseDragged(MouseEvent me)
{
}

public void paint(Garphics g)
{
     g.drawString(msg, mousex, mousey);
}
}
```

Handling KeyEvent



Adapter Class

- Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular eventlistener interface.
- You can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which you are interested.
- If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it *saves code*.

Contd…

- For example, the MouseMotionAdapter class has two methods, mouseDragged( ) and mouseMoved( ).
- The signatures of these empty methods are exactly as defined in the MouseMotionListener interface.
- If you are interested in only mouse drag events, then you could simply extend MouseMotionAdapter and can implement mouseDragged( ).
- The empty implementation of mouseMoved( ) would handle the mouse motion events for you.

JAVA.AWT.EVENT ADAPTER CLASSES

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

Contd…

- Internally Adapter class implements listener interfaces.
- For example examine the code below:

```
interface MouseListener
{
    public void mouseClicked(MouseEvent me);
    public void mouseEntered(MouseEvent me);
    public void mouseExited(MouseEvent me);
    public void mousePressed(MouseEvent me);
    public void mouseReleased(MouseEvent me);
 }
```

- The problem with interfaces is that even we have to handle only one event say mouseClicked, we will have to implement all the methods present in the above interface.
- A limitation with the interfaces is if you implement them, you have to implement all the methods available in them. You can't leave any one!

Contd…

- A savior for this is our Adapter class.
- For the MouseListener interface we have a corresponding Adapter class called as MouseAdapter which has the following definition:

```
class MouseAdapter implements MouseListener
{
    public void mouseClicked(MouseEvent me)
      {
      }
     public void mouseEntered(MouseEvent me)
      {
     }
     public void mouseExited(MouseEvent me)
      {
      }
     public void mousePressed(MouseEvent me)
      {
      }
     public void mouseReleased(MouseEvent me)
      {
       }
}
```

Contd…

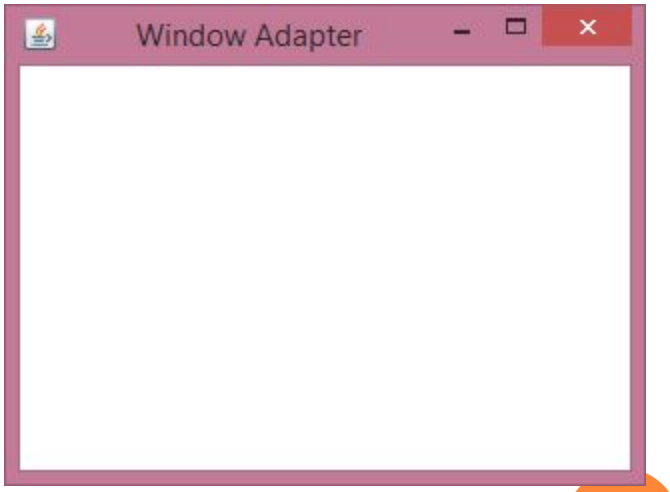
- In the above example, the MouseAdapter class implements MouseListener and gives an empty definitions for all the methods available in them.
- All you have to do is extend the MouseAdapter class and override any of the methods you want to but not all of them.
- **Syntax:**

```
public void class A extends MouseAdapter
{
    public void mouseClicked(MouseEvent m)
     {
        System.out.println("Mouse Clicked");
     }
}
```

Java WindowAdapter Example

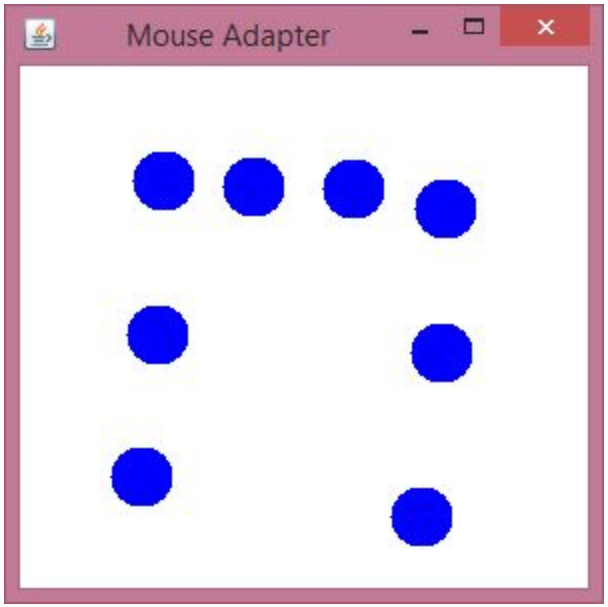
```
import java.awt.*;
import java.awt.event.*;
public class AdapterExample{
   Frame f;
   AdapterExample(){
      f=new Frame("Window Adapter");
      f.addWindowListener(new WindowAdapter(){
         public void windowClosing(WindowEvent e) {
            f.dispose();
         }
      });

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

Output:



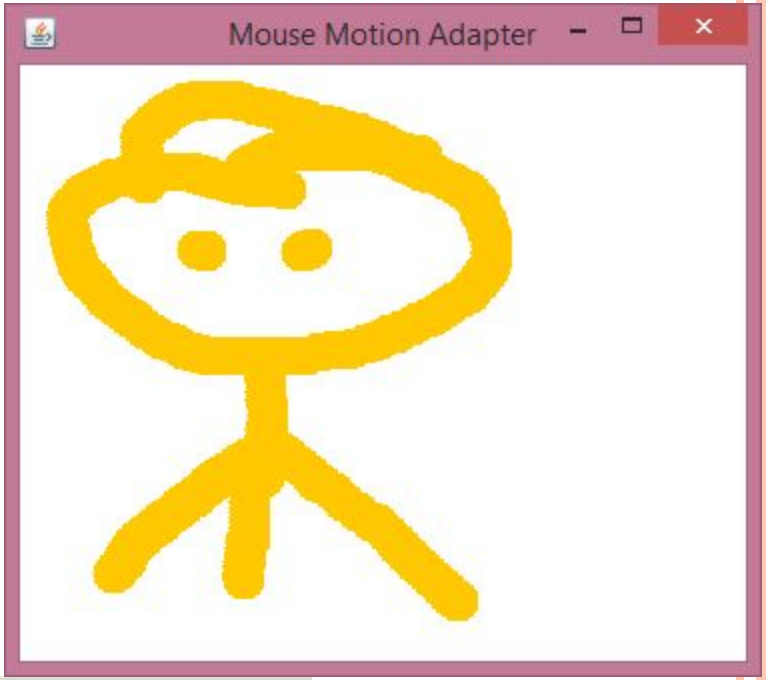
Java MouseAdapter Example

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

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

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



Java MouseMotionAdapter Example

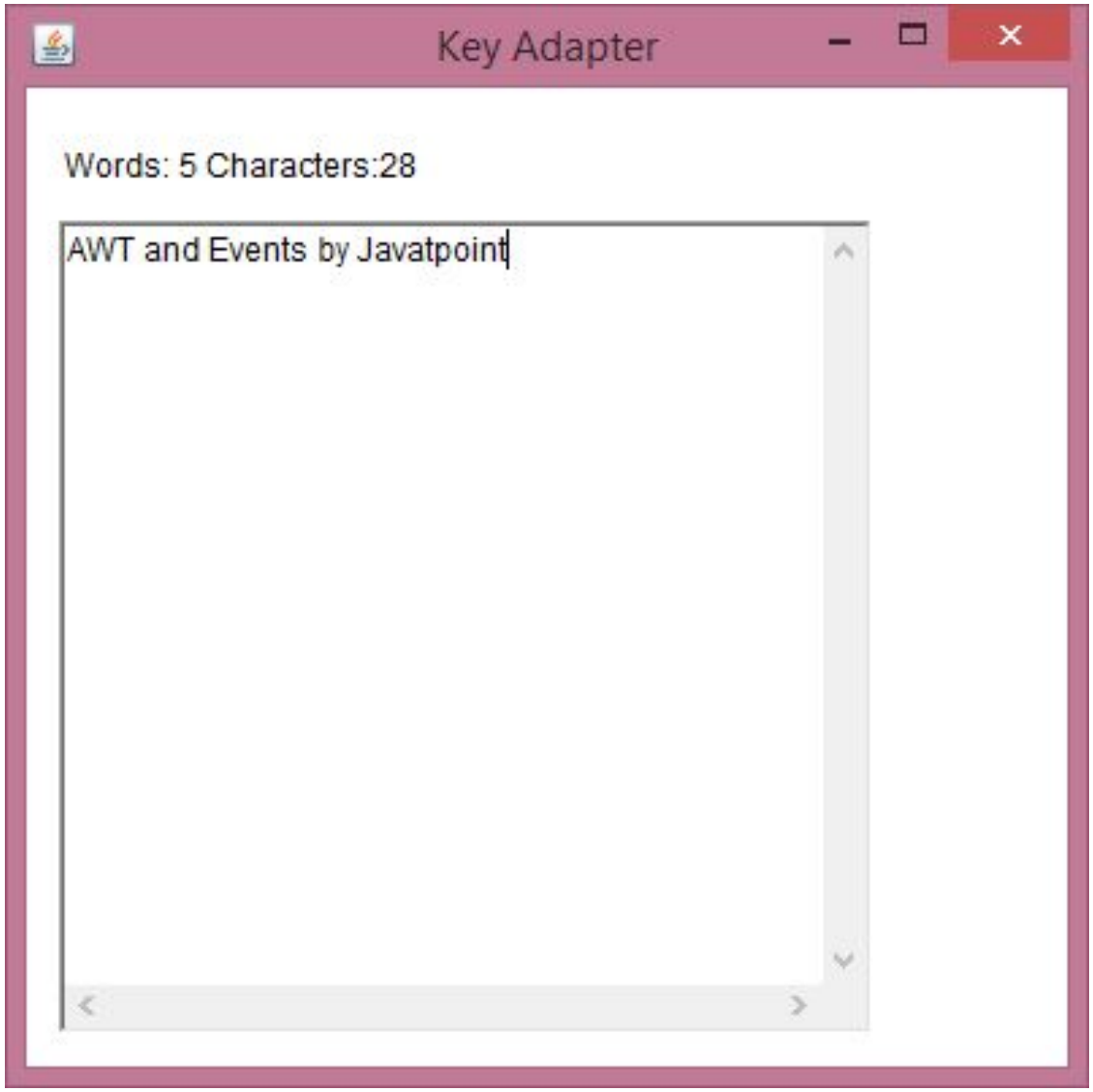
```
import java.awt.*;
import java.awt.event.*;
public class MouseMotionAdapterExample extends MouseMotionAdapter{
    Frame f;
    MouseMotionAdapterExample(){
        f=new Frame("Mouse Motion Adapter");
        f.addMouseMotionListener(this);

        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
public void mouseDragged(MouseEvent e) {
    Graphics g=f.getGraphics();
    g.setColor(Color.ORANGE);
    g.fillOval(e.getX(),e.getY(),20,20);
}
public static void main(String[] args) {
    new MouseMotionAdapterExample();
}
}
```



Java KeyAdapter Example

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

        f.add(l);f.add(area);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }

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

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

CONTD…


Key Adapter
Words: 5 Characters:28
AWT and Events by Javatpoint

Inner class

- Inner class is a class defined within other class, or even within an expression. This section illustrates how inner classes can be used to simplify the code when using event adapter classes.
- To understand the benefit provided by inner classes, consider the applet shown in the following listing. It does not use an inner class.
- Its goal is to display the string “Mouse Pressed” in the status bar of the applet viewer or browser when the mouse is pressed.
- There are two top-level classes in this program. MousePressedDemo extends Applet, and MyMouseAdapter extends MouseAdapter.
- The init( ) method of MousePressedDemo instantiates MyMouseAdapter and provides this object as an argument to the addMouseListener( ) method.

Contd…

- Notice that a reference to the applet is supplied as an argument to the MyMouseAdapter constructor.
- This reference is stored in an instance variable for later use by the mousePressed( ) method. When the mouse is pressed, it invokes the showStatus( ) method of the applet through the stored applet reference.
- In other words, showStatus( ) is invoked relative to the applet reference stored by MyMouseAdapter.

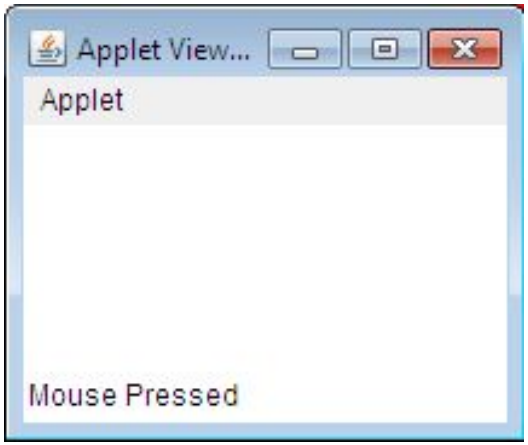
Example without inner class

InnerClassDemo - Notepad

File Edit Format View Help

```
import java.applet.*;
import java.awt.event.*;
/*
<applet code="InnerClassDemo" width=200 height=100>
</applet>
*/
public class InnerClassDemo extends Applet {
public void init() {
addMouseListener(new MyMouseAdapter());
}
class MyMouseAdapter extends MouseAdapter {
public void mousePressed(MouseEvent me) {
showStatus("Mouse Pressed");
}
}
}
```

Output-



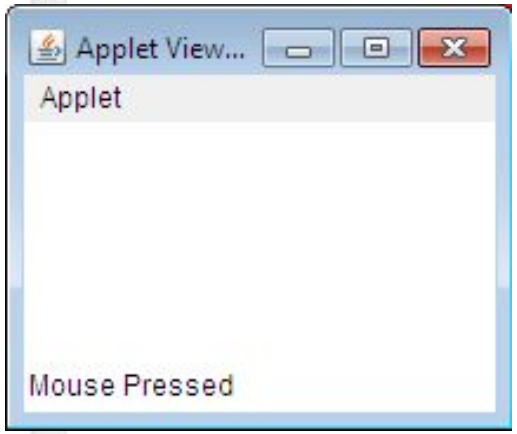
Anonymous Inner Classes

- An anonymous inner class is one that is not assigned a name.
- **Example-**

AnonymousInnerClassDemo - Notepad

File Edit Format View Help

```
import java.applet.*;
import java.awt.event.*;
/*<applet code="AnonymousInnerClassDemo" width=200
height=100></applet>*/


public class AnonymousInnerClassDemo extends Applet {
public void init() {
addMouseListener(new MouseAdapter() {
public void mousePressed(MouseEvent me) {
showStatus("Mouse Pressed");
}
});
}
```



Contd…

- There is one top-level class in this program:
  - AnonymousInnerClassDemo. The init( ) method calls the addMouseListener( ) method.
  - The syntax new MouseAdapter( ) { ... } indicates to the compiler that the code between the braces defines an anonymous inner class.
  - Furthermore, that class extends MouseAdapter.
  - This new class is not named, but it is automatically instantiated when this expression is executed.
  - Because this anonymous inner class is defined within the scope of AnonymousInnerClassDemo, it has access to all of the variables and methods within the scope of that class.
  - Therefore, it can call the showStatus( ) method directly

Introduction to Swings

- **Swing** is a set of classes that provides **more powerful** and flexible components than are possible with the **AWT**.
- In addition to the familiar components, such as buttons, check boxes, and labels, Swing supplies several exciting additions, including tabbed panes, scroll panes, trees, and tables.
- Even familiar components such as buttons have more capabilities in Swing.
- For example, a button may have both an image and a text string associated with it. Also, the image can be changed as the state of the button changes.

Swing Features

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

Containers

- Swing defines two types of containers.
  - Top-Level Containers.
  - LightWeight Containers.
- ***The top-Level Containers:***
- The first are top-level containers are **JFrame**, **JApplet**, **JWindow**, and **JDialog**.
- These containers do not inherit **JComponent**.
- They do, however, inherit the AWT classes **Component** and **Container**.
- The top-level containers are **heavyweight.**
- A top-level container is not contained within any other container.
- The one most commonly used for applications is **JFrame**.
- The one used for applets is **JApplet**.

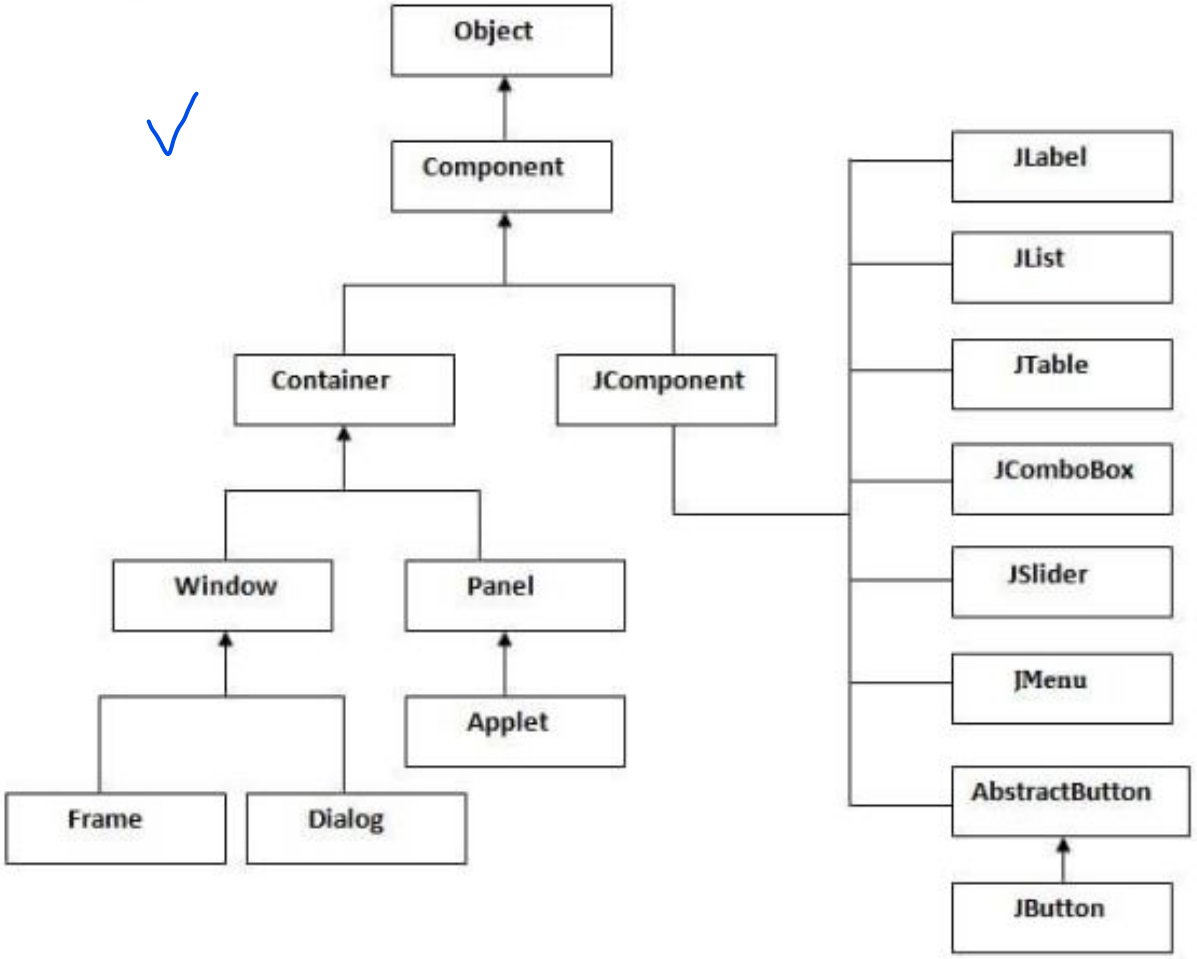
CONTD….

- ***The LightWeight Containers:***
- The second type of containers supported by Swing are **lightweight** containers.
- **Lightweight** containers *do* inherit **Jcomponent.**
- An example of a lightweight container is **Jpanel**, which is a general-purpose container.
- **Lightweight** containers are often used to organize and manage groups of related components because a **lightweight** container can be contained within another container.

Difference between AWT and Swing

Java AWT	Java Swing
AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
AWT components are **heavyweight**.	Swing components are **lightweight**.
AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

Hierarchy of Java Swing classes



The Swing-related classes are contained in **javax.swing** and its subpackages, such as **javax.swing.tree.**

SWING UI ELEMENTS

S.No.	Class & Description
1	JLabel-A JLabel object is a component for placing text in a container.
2	JButton- This class creates a labeled button.
3	JColorChooser- A JColorChooser provides a pane of controls designed to allow a user to manipulate and select a color.
4	JCheckBox-A JCheckBox is a graphical component that can be in either an **on** (true) or **off** (false) state.
5	JRadioButton-The JRadioButton class is a graphical component that can be in either an **on** (true) or **off** (false) state. in a group.
6	JList- A JList component presents the user with a scrolling list of text items.
7	JComboBox- A JComboBox component presents the user with a to show up menu of choices.
8	JTextField- A JTextField object is a text component that allows for the editing of a single line of text.

CONTD…

9	**JPasswordField**- A JPasswordField object is a text component specialized for password entry.
10	**JTextArea**- A JTextArea object is a text component that allows editing of a multiple lines of text.
11	**ImageIcon**- A ImageIcon control is an implementation of the Icon interface that paints Icons from Images
12	**JScrollbar**- A Scrollbar control represents a scroll bar component in order to enable the user to select from range of values.
13	**JOptionPane**- JOptionPane provides set of standard dialog boxes that prompt users for a value or informs them of something.
14	**JFileChooser**- A JFileChooser control represents a dialog window from which the user can select a file.
15	**JProgressBar**- As the task progresses towards completion, the progress bar displays the task's percentage of completion.
16	**Jslider**- A JSlider lets the user graphically select a value by sliding a knob within a bounded interval.
17	**Jspinner**- A JSpinner is a single line input field that lets the user select a number or an object value from an ordered sequence.

Types of Swing Programs

- Desktop Application Programs
- Applet Programs

Desktop application example1

```
import java.awt.event.*;
import javax.swing.*;
public class Button1 {
public static void main(String[] args) {
    JFrame f=new JFrame("Button Example");
    final JTextField tf=new JTextField();
    tf.setBounds(50,50, 150,20);
    JButton b=new JButton("Click Here");
    b.setBounds(50,100,95,30);
    b.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent e){
            tf.setText("Welcome to Java.");
        }
    });
    f.add(b);f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Button component through swings

Another example using event handling

```
import java.awt.event.*;
import javax.swing.*;
public class Button1  implements ActionListener {
 JTextField tf1,tf2,tf3;
 JButton b1,b2;
 Button1(){
 JFrame f= new JFrame();
 tf1=new JTextField();
 tf1.setBounds(50,50,150,20);
 tf2=new JTextField();
 tf2.setBounds(50,100,150,20);
 tf3=new JTextField();
 tf3.setBounds(50,150,150,20);
 tf3.setEditable(false);
 b1=new JButton("+");
 b1.setBounds(50,200,50,50);
 b2=new JButton("-");
 b2.setBounds(120,200,50,50);
 b1.addActionListener(this);
 b2.addActionListener(this);
 f.add(tf1);
 f.add(tf2);
 f.add(tf3);
 f.add(b1);
 f.add(b2);
 f.setSize(300,300);
 f.setLayout(null);
 f.setVisible(true); }
 public void actionPerformed(ActionEvent e){
 String s1=tf1.getText();
 String s2=tf2.getText();
 int a=Integer.parseInt(s1);
 int b=Integer.parseInt(s2);
 int c=0;
 if(e.getSource()==b1){
 c=a+b; }
else if(e.getSource()==b2)   {
   c=a-b;  }
 String result=String.valueOf(c);
 tf3.setText(result);  }
public static void main(String[] args){
 new Button1();
}}
```

Create a Swing Applet using JApplet

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

EXAMPLE-

```
import java.applet.*;
import javax.swing.*;
import java.awt.event.*;
public class EventJApplet extends JApplet implements ActionListener{
JButton b;
JTextField tf;
public void init(){

tf=new JTextField();
tf.setBounds(30,40,150,20);

b=new JButton("Click");
b.setBounds(80,150,70,40);

add(b);add(tf);
b.addActionListener(this);

setLayout(null);
}

public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
}
```

myapplet.html

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