Applet:

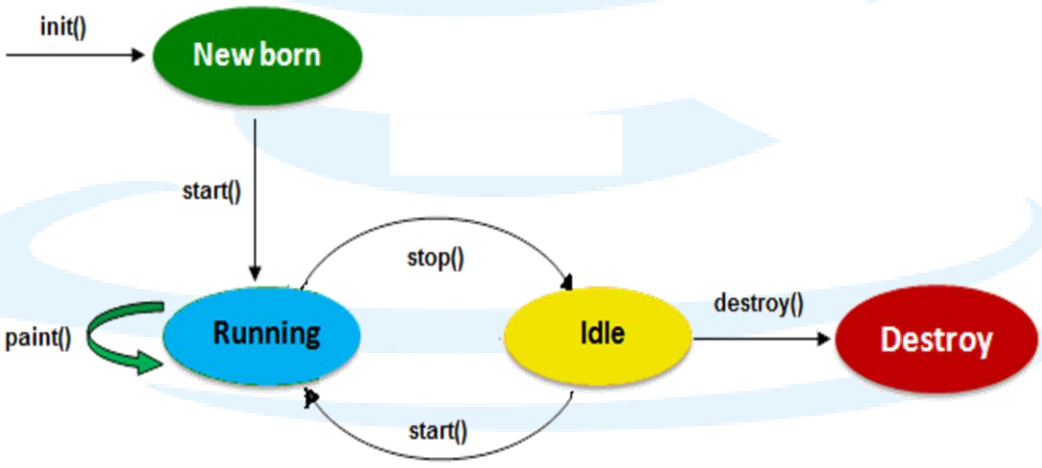
When a HTML (Hyper Text Markup Language) page wants to communicate with the user on Internet, it can use a special Java program, called "applet" to communicate and respond to the user. The user can interact by typing some details or by clicking the components available in the applet program. The program then processes and displays the results.

Applet = Java Byte Code + HTML

Life Cycle of an Applet

Four methods in the Applet class gives you the framework on which you build any serious applet −

- **public void init()** – This method is the first method to be called by the browser and it is executed only once. So, the programmer should use this method to initialize any variables, creating components and creating threads, etc.
- **public void start()** − This method is called after init() method and each time the applet is revisited by the user. For example, the user minimized and the webpage that contains the applet is moved to another page then this method's execution is stopped. Any calculations and processing of data should be done in this method and result is displayed.
- **public void stop()** − This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- **public void destroy()** − This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.



Creation of applet:

To create an applet, we need to .write a Java program and compile it to get byte code. Then we should embed (include) it into a HTML page on a particular location wherever we want it to be displayed. This page is then stored in the web server. A client machine communicates with the web server, the server then sends the HTML page that contains the applet. The page is then transmitted to the client where the applet is executed on the client's web browser. Thus applets are executed at client side by the web browser. To create an applet, we have Applet class of **java.applet** package.

Uses of Applets

- Applets are used on Internet for creating dynamic web pages. There are two types of web pages: Static and Dynamic. Static web pages provide some information to the user but the user cannot interact with the webpage other than viewing the information. Dynamic web-pages interact with the user at the runtime. For example, a student can type his hall ticket number in a-text field and click the retrieve button to get back his results from his University server. Applets are useful to provide such interaction with the user at runtime.
- Another use of applet is for creating animation and games where the images can be displayed or moved giving a visual impression that they are alive.

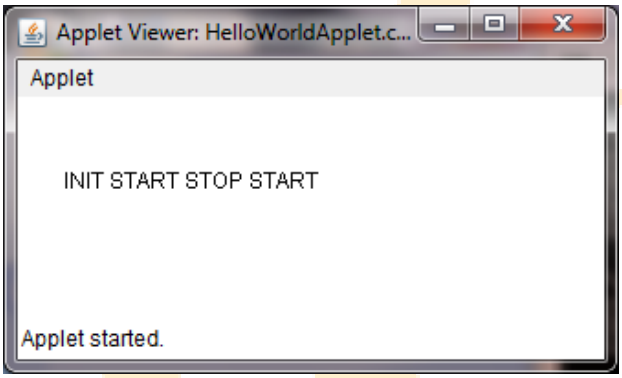
A Simple Applet Program:

```
import java.applet.*;
import java.awt.*;
public class MyApplet extends Applet
{
      String msg="";
      public void init()
      {
            msg+="INIT ";
      }
      public void start()
      {
            msg+="START ";
      }
      public void stop()
      {
            msg+="STOP ";
      }
      public void destroy()
      {
            msg+="DESTROY ";
      }
   public void paint (Graphics g)
   {
      g.drawString (msg, 25, 50);
   }
}
```

Save the above program with **MyApplet.java** and compile the program. There will be a **MyApplet.class** file will be generated. This class file will be included into html file. And the html file will be written as:

```
<html>
<applet code= “MyApplet.class” width= “400” height= “500”></applet>
<html>
```

Save the above html file **MyApplet.html.** To run this html file we should type the following command: **appletviewer MyApplet.html**

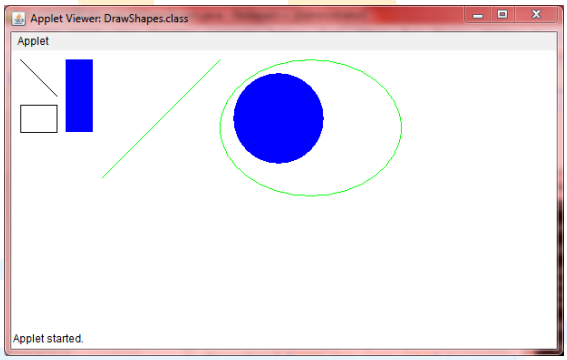


Methods in Graphics Class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **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.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **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.
6. **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).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **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.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

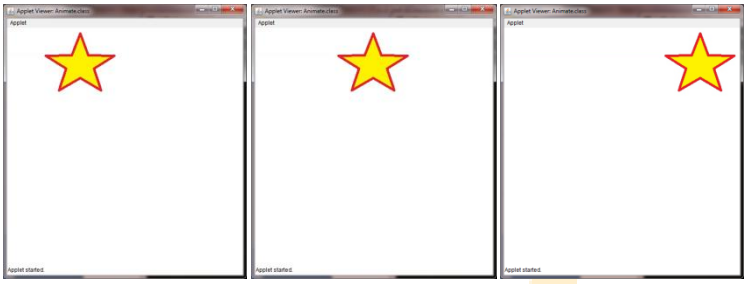
Example: Write a JAVA program to create different shapes and fill colors using Applet.

```
import java.awt.*;
import java.applet.*;
public class DrawShapes extends Applet
{
      public void paint(Graphics g)
      {
        g.drawLine(10,10,50,50);
        g.drawRect(10,60,40,30);
        g.setColor(Color.blue);
        g.fillRect(60,10,30,80);
        g.setColor(Color.green);
        g.drawLine(100,140,230,10);
        g.drawOval(230,10,200,150);
        g.setColor(Color.blue);
        g.fillOval(245,25,100,100);
      }
}
```



Example: Write a Java Program to display moving of an image.

```
import java.applet.*;
import java.awt.*;
public class Animate extends Applet
{
   public void paint (Graphics g)
  {
        Image img=getImage(getDocumentBase(),"hello.jpg");
        for(int i=0;i<800;i++)
        {
              g.drawImage(img,i,0,null);
              try{
                    Thread.sleep(20);
              }catch(Exception e){}
        }
   }
}
```



What is an Event?

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

Types of Event:

The events can be broadly classified into two categories −

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

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

What is Event Handling?

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

Java Delegation Model:

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

The Delegation Event Model has the following key participants.

Source − the source is an object on which the event occurs. Source is responsible for providing information of the occurred event to its handler. Java provides us with classes for the source object.

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

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

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

Steps Involved in Event Handling

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

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

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

Step 4 − The method is gets executed and returns.

Points to Remember About the Listener

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

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

Event and Listener (Java Event Handling):

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

Java Event classes and Listener interfaces:

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

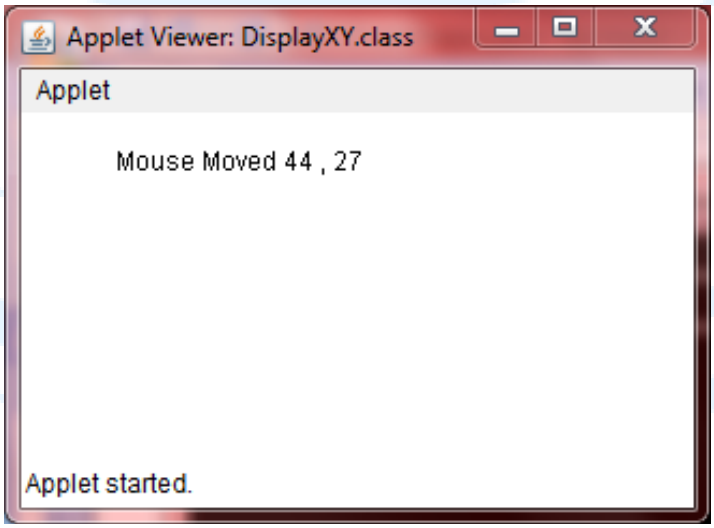
Listeners and Methods:

Listener Interfaces	Methods
ActionListener	public void actionPerformed(ActionEvent ae)
MouseListener	public void mouseClicked(MouseEvent e) public void mouseEntered(MouseEvent e) public void mouseExited(MouseEvent e) public void mousePressed(MouseEvent e) public void mouseReleased(MouseEvent e)
MouseMotionListener	public void mouseDragged(MouseEvent e) public void mouseMoved(MouseEvent e)
MouseWheelListener	public void mouseWheelMoved(MouseWheelEvent)
KeyListener	public void keyPressed(KeyEvent e) public void keyReleased(KeyEvent e) public void keyTyped(KeyEvent e)
ItemListener	public void itemStateChanged(ItemEvent e)
TextListener	public void textValueChanged(TextEvent e)
AdjustmentListener	public void adjustmentValueChanged(AdjustmentEvent e)
WindowListener	public void windowActivated(WindowEvent e) public void windowClosed(WindowEvent e) public void windowClosing(WindowEvent e) public void windowDeactivated(WindowEvent e) public void windowDeiconified(WindowEvent e) public void windowIconified(WindowEvent e) public void windowOpened(WindowEvent e)
ComponentListener	public void componentHidden(ComponentEvent e) public void componentMoved(ComponentEvent e) public void componentResized(ComponentEvent e) public void componentShown(ComponentEvent e)
ContainerListener	public void componentAdded(ContainerEvent e) public void componentRemoved(ContainerEvent e)
FocusListener	public void focusGained(FocusEvent e) public void focusLost(FocusEvent e)

Example: Write a JAVA program that display the x and y position of the cursor movement using Mouse.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class DisplayXY extends Applet implements MouseMotionListener
{
      int x,y;
      String str="";
      public void init()
      {
            addMouseMotionListener(this);
      }
      public void mouseDragged(MouseEvent me)
      {
             x = me.getX();
             y = me.getY();
             str = "Mouse Dragged "+x+" , "+y;
             repaint();
      }
      public void mouseMoved(MouseEvent me)
      {
             x = me.getX();
             y = me.getY();
             str = "Mouse Moved "+x+" , "+y;
             repaint();
      }
      public void paint(Graphics g)
      {
           g.drawString(str, x, y);
      }
}
```

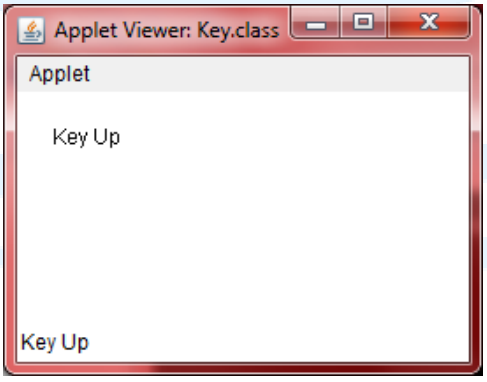
Output:



Example: Write a JAVA program that identifies key-up key-down event user entering text in a Applet.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class Key extends Applet implements KeyListener
{
      int X=20,Y=30;
      String msg="";
      public void init()
      {
            addKeyListener(this);
            requestFocus();
      }
      public void keyPressed(KeyEvent k)
      {
            showStatus("Key Pressed");
            msg="Key Pressed";
            repaint();
      }
      public void keyReleased(KeyEvent k)
      {
            showStatus("Key Up");
            msg="Key Up";
            repaint();
      }
      public void keyTyped(KeyEvent k)
      {
            showStatus("Key Typed");
            msg="Key Typed";
            repaint();
      }
      public void paint(Graphics g)
      {
            g.drawString(msg,X,Y);
      }
}
```

Output:



Example: Write a JAVA program that can perform WindowEvents.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class WindowEventsDemo extends Applet implements WindowListener
{
	String msg="";
	public void init()
	{
		addWindowListener(this);
	}
	public void windowActivated(WindowEvent we)
	{
		msg="Window Activated";
		repaint();
	}
	public void windowDeactivated(WindowEvent we)
	{
		msg="Window Deactivated";
		repaint();
	}
	public void windowOpened(WindowEvent we)
	{
		msg="Window Opened";
		repaint();
	}
	public void windowClosed(WindowEvent we)
	{
		msg="Window Closed";
		repaint();
	}
	public void windowClosing(WindowEvent we)
	{
		msg="Window Closing";
		repaint();
		System.exit(0);
	}
	public void windowIconified(WindowEvent we)
	{
		msg="Window Iconified";
		repaint();
	}
	public void windowDeiconified(WindowEvent we)
	{
		msg="Window Deiconified";
		repaint();
```

```
	}
	public void paint(Graphics g)
	{
		g.drawString(msg,10,20);
	}
}
```

Adapter classes:

The Adapter class provides the default modification of all methods of an interface; we don't need to modify all the methods of the interface so we can say it reduces coding burden. Sometimes or often we need a few methods of an interface. For that the Adapter class is very helpful since it already modifies all the methods of an interface and by implementing the Adapter class, we only need to modify the required methods.

Examples:

The following examples contain the following Adapter classes:

- ContainerAdapter class
- KeyAdapter class
- FocusAdapter class
- WindowAdapter class
- MouseAdapter class
- ComponentAdapter class
- MouseMotionAdapter class

Inner classes:

In Java, just like methods, variables of a class too can have another class as its member. Writing a class within another is allowed in Java. The class written within is called the nested class, and the class that holds the inner class is called the outer class.

Syntax:

```
class Outer_Demo
{
	class Nested_Demo
	{
	}
}
```

Note: Refer Java 2nd Unit Notes