UNIT - 5

APPLETS

EVENT HANDLING

Applets

Definition of Applet

An applet is a small Java program that is a part of a web page and is downloaded automatically from a server when the client requests that web page.

Applet Fundamentals

Since applet is a part of a webpage, we will learn basic things related to web pages. A web page is a collection of information (ex: Google home page, Yahoo home page etc). Web pages are generally created using HTML.

A web page contains at least four basic HTML tags:

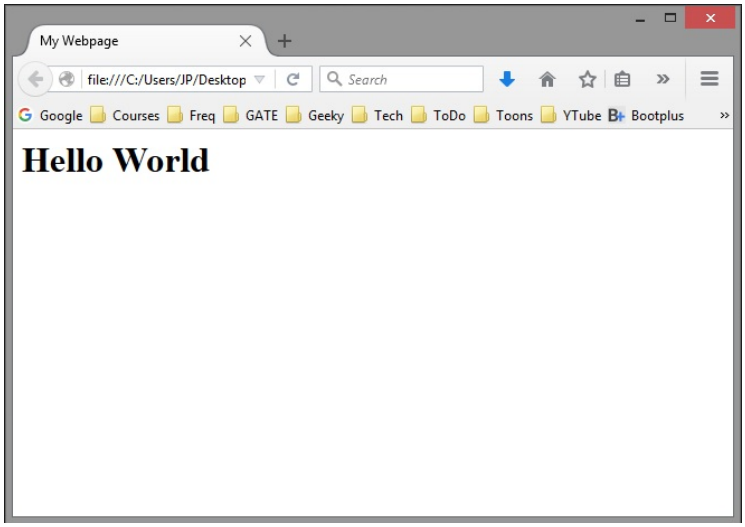
- **<html> :** Specifies that it is the root tag.
- **<head> :** Specifies the head section of the web page. It contains meta information, page title, scripts, external style sheet links etc.
- **<body> :** Contains the content that is displayed to a user who is visiting the web page.
- **<title> :** Specifies the title of the web page. It is specified inside <head> tags.

As an example let's look at the HTML code of a web page that displays "Hello World" in large letters. The code is as follows:

hello.html

```
<html>
    <head>
        <title>My Webpage</title>
    </head>
    <body>
        <h1>Hello World</h1>
    </body>
</html>
```

The file will be saved with .html extension. When you open the file with a web browser, the output will be as follows:



An applet can be displayed in a web page using the <applet> tag as shown below:

```
<applet code="ClassName" height="200" width="400"></applet>
```

An applet doesn't contain a main() method like the Java console programs. An applet is used to display graphics or to display a GUI (ex: login form, registration form) to the users.

Creating an Applet

An applet can be created by extending either Applet class or JApplet class. First let's see how to create an applet using the Applet class. The Applet class is available in java.applet package. An applet which displays "Hello World" is shown below:

MyApplet.java

```
import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet
{
	public void paint(Graphics g)
	{
		g.drawString("Hello World!", 20, 20);
	}
}
/*
	<applet code="MyApplet" height="300" width="500"></applet>
*/
```

Remember to include the <applet> tag in comments. This is useful for running the applet. The value of code attribute must match with the class name.

In the above applet program, the class MyApplet extends the Applet class and it contains a method named paint() which accepts a parameter of the type Graphics. The Graphics class belongs to java.awt package and is used to display text or graphics on our applet.

The paint() method is used to re-display the output when the applet is resized or minimized or maximized. The drawString() method belongs to Graphics class and it displays "Hello World!" on the applet display area at the point whose coordinates are (20, 20).

Save the file as MyApplet.java and compile it to generate MyApplet.class file.

Running an Applet

An applet can be executed in two ways:

- Using appletviewer command-line tool or
- Using a browser

Using appletviewer tool:

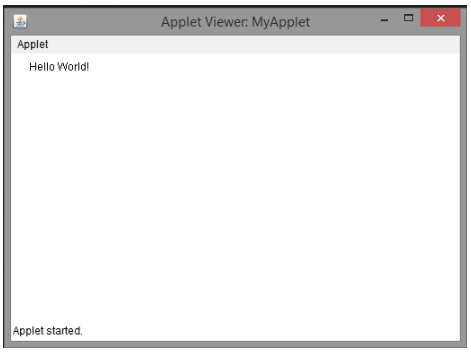
Java provides a command line tool named appletviewer for quick debugging of applets. Use the following syntax for running an applet using appletviewer:

```
appletviewer filename.java
```

As the file name of our applet is MyApplet.java, the command for executing the applet is:

```
appletviewer MyApplet.java
```

Output of the above command will be as shown below:



Using a browser:

Create a web page named hello.html with the following HTML code:

```
<html>
    <head>
        <title>My Webpage</title>
    </head>
    <body>
        <applet code="MyApplet" height="200" width="400"></applet>
    </body>
</html>
```

Unfortunately from Java 7 onwards, local applets (unsigned) cannot be viewed using a browser and Google Chrome browser no longer supports applets in web pages.

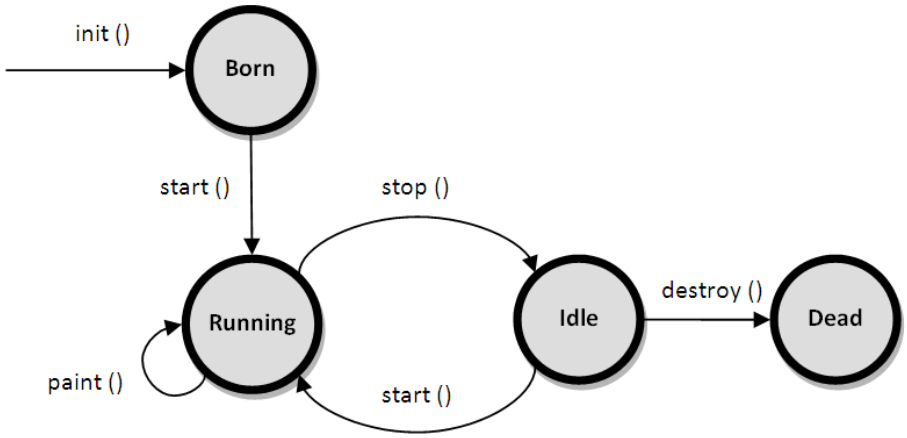
It is recommended to use appletviewer tool to debug and execute applets.

Differences between a Java Application and Applets

Java Application	Applet
Java application contains a main method	An applet does not contain a main method
Does not require internet connection to execute	Requires internet connection to execute
Is stand alone application	Is a part of web page
Can be run without a browser	Requires a Java compatible browser
Uses stream I/O classes	Use GUI interface provided by AWT or Swings
Entry point is main method	Entry point is init method
Generally used for console programs	Generally used for GUI interfaces

Applet Life Cycle

The life cycle of an applet is as shown in the figure below:



As shown in the above diagram, the life cycle of an applet starts with init() method and ends with destroy() method. Other life cycle methods are start(), stop() and paint(). The methods to execute only once in the applet life cycle are init() and destroy(). Other methods execute multiple times.

Below is the description of each applet life cycle method:

init(): The init() method is the first method to execute when the applet is executed. Variable declaration and initialization operations are performed in this method.

start(): The start() method contains the actual code of the applet that should run. The start() method executes immediately after the init() method. It also executes whenever the applet is restored, maximized or moving from one tab to another tab in the browser.

stop(): The stop() method stops the execution of the applet. The stop() method executes when the applet is minimized or when moving from one tab to another in the browser.

destroy(): The destroy() method executes when the applet window is closed or when the tab containing the webpage is closed. stop() method executes just before when destroy() method is invoked. The destroy() method removes the applet object from memory.

paint(): The paint() method is used to redraw the output on the applet display area. The paint() method executes after the execution of start() method and whenever the applet or browser is resized.

The method execution sequence when an applet is executed is:

- init()
- start()
- paint()

The method execution sequence when an applet is closed is:

- stop()
- destroy()

Example program that demonstrates the life cycle of an applet is as follows:

```
import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet
{
    public void init()
    {
        System.out.println("Applet initialized");
```

```
    }
    public void start()
    {
        System.out.println("Applet execution started");
    }
    public void stop()
    {
        System.out.println("Applet execution stopped");
    }
    public void paint(Graphics g)
    {
        System.out.println("Painting...");
    }
    public void destroy()
    {
        System.out.println("Applet destroyed");
    }
}
```

Output of the above applet program when run using appletviewer tool is:

```
Applet initialized
Applet execution started
Painting…
Painting…
Applet execution stopped
Applet destroyed
```

Applet Class

The Applet class provides the support for creation and execution of applets. It allows us to start and stop the execution of an applet. Applet class extends the Panel class which extends Container class and which in turn extends the Component class. So applet supports all window-based activities.

Methods in Applet Class

Below are some of the methods available in the class Applet:

init() – This method is called when the execution of the applet begins. It is the entry point for all applets.

start() – This method is automatically executed after the init() method completes its execution or whenever the execution of an applet is resumed.

stop() – This method executes whenever the execution of the applet is suspended.

destroy() – This method is called when the webpage containing the applet is closed.

getCodeBase() – Returns the URL associated with the applet.

getDocumentBase() – Returns the URL of the HTML document that invokes the applet.

getParameter(String paramName) – Returns the value associated with the paramName. Returns null if there is no value.

isActive() – Returns true if the applet is started or false if the applet is stopped.

resize(int width, int height) – Resizes the applet based on the given width and height.

showStatus(String str) – Displays the string str in the status bar of the browser or applet.

Passing Parameters to Applets

Parameters specify extra information that can be passed to an applet from the HTML page. Parameters are specified using the HTML's param tag.

Param Tag

The <param> tag is a sub tag of the <applet> tag. The <param> tag contains two attributes: name and value which are used to specify the name of the parameter and the value of the parameter respectively. For example, the param tags for passing name and age parameters looks as shown below:

```
<param name="name" value="Ramesh" />
<param name="age" value="25" />
```

Now, these two parameters can be accessed in the applet program using the getParameter() method of the Applet class.

getParameter() Method

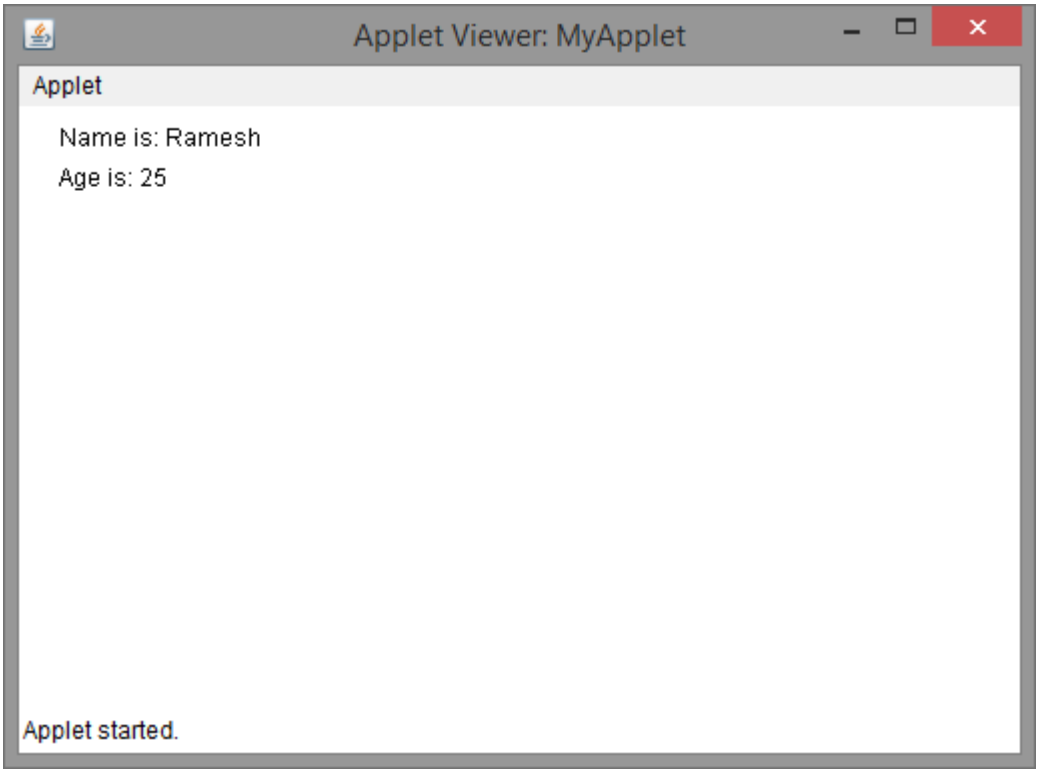
The getParameter() method of the Applet class can be used to retrieve the parameters passed from the HTML page. The syntax of getParameter() method is as follows:

```
String getParameter(String param-name)
```

Let's look at a sample program which demonstrates the <param> HTML tag and the getParameter() method:

```
import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet
{
	String n;
	String a;
	public void init()
	{
		n = getParameter("name");
		a = getParameter("age");
	}
	public void paint(Graphics g)
	{
		g.drawString("Name is: " + n, 20, 20);
		g.drawString("Age is: " + a, 20, 40);
	}
}
/*
	<applet code="MyApplet" height="300" width="500">
		<param name="name" value="Ramesh" />
		<param name="age" value="25" />
	</applet>
*/
```

Output of the above program is as follows:



Applet Tag

applet Tag Attributes

Following are the various attributes of applet tag:

- **code**: Used to specify the name of the applet class.
- **codebase**: Used to specify the location at which the applet code (class) is available.
- **width**: Used to specify the width of the applet display area.
- **height**: Used to specify the height of the applet display area.
- **name**: Used to specify the name of the applet which is helpful in distinguish one applet from another when there are multiple applets in the same webpage.

- **hspace**: Used to specify the horizontal space to the left and right of the applet display area.
- **vspace**: Used to specify the vertical space to the top and bottom of the applet display area.
- **align**: Used to specify the alignment of the applet display area with respect to other HTML elements.

Above listed attributes are the most frequently used attributes of the applet tag. Below is sample HTML code using some of the attributes specified above.

```
<applet code="MyApplet" codebase="http://www.abc.com/applets/"
height="300" width="400" name="applet1" hspace="20"
vspace="20">
</applet>
```

Event Handling

Definition: The process of handling events is known as event handling. Java uses "delegation event model" to process the events raised in a GUI program.

Delegation Event Model

The delegation event model provides a consistent way of generating and processing events. In this model a source generates an event and sends it to one or more listeners. A listener waits until it receives an event and once it receives an event, processes the event and returns.

The advantage of this model is the application logic for processing the events is clearly separated from the user interface logic. In this model the listeners must register with a source to receive the events.Another advantage of this model is, notification of events is sent only to the registered listeners.

Events

An event is an object that describes a state change in the source. Some examples of activities that cause events to be generated are clicking a button, pressing a key on the keyboard, selecting an item in the list.

Event Sources

A source is an object that generates an event. A source might generate more than one event. Examples of sources are button, check box, radio button, text box etc.

A source must be registered with listeners in order for the events to be detected. The general form of a registration method is as follows:

public void add*Type*Listener(*Type*Listener tl)

A listener can also be unregistered from the source using the remove method whose syntax is as follows:

public void remove*Type*Listener(*Type*Listener tl)

Event Listeners

A listener is an object that is notified when an event occurs. A listener has two major requirements. First is, listener must be registered with a source to receive notifications and second is, it must implement methods to receive and process these notifications.

In Java, events and sources are maintained as classes, listeners are maintained as interfaces. Most of them are available in java.awt.event package. Examples of listeners are MouseListener, Key Listener etc.

java.awt.event Package

The java.awt.event package contains many event classes which can be used for event handling. Root class for all the event classes in Java is EventObject which is available in java.util package. Root class for all AWT event classes is AWTEvent which is available in java.awt package and is a sub class of EventObject class.

Some of the frequently used event classes available in java.awt.event package are listed below:

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 superclass 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.
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.
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.

Various constructors and methods available in the above listed event classes are as follows:

ActionEvent

An action event is generated when a button is clicked, or a list item is double-clicked, or a menu item is selected.

ActionEvent class contains the following methods:

String getActionCommand() – Used to obtain the command name for the invoking ActionEvent object.

int getModifiers() – This method returns an integer that indicates which modifier keys (ALT, CTRL, META, and/or SHIFT) were pressed.

long getWhen() – Returns the time when the event was generated.

AdjustmentEvent

The adjustment event is generated when the scroll bar is adjusted. There are five types of adjustment events. For each type there is a constant declared in the class which are as follows:

BLOCK_DECREMENT
BLOCK_INCREMENT
TRACK
UNIT_DECREMENT
UNIT_INCREMENT

Following are the methods available in AdjustmentEvent class:

Adjustable getAdjustable() – Returns the object that generated the event.

int getAdjustmentType() – Returns one of the five constants that represents the type of event.

int getValue() – Returns a number that represents the amount of adjustment.

ComponentEvent

A component event is generated when the visibility, position and size of a component is changed. There are four types of component events which are identified by the following constants:

COMPONENT_HIDDEN
COMPONENT_SHOWN
COMPONENT_MOVED
COMPONENT_RESIZED

Following method is available in the ComponentEvent class:

Component getComponent() – Returns the component that generated the event.

ContainerEvent

A container event is generated when a component is added to or removed from the container. There are two types of container events which are represented by the following constants:

```
COMPONENT_ADDED
COMPONENT_REMOVED
```

Following are the methods available in the ContainerEvent class:

Container getContainer() – Returns the reference of the container that generated the event.

Component getChild() – Returns the reference of the component that is added to or removed from the container.

FocusEvent

A focus event is generated when a component gains or loses input focus. These events are identified by the following constants:

```
FOCUS_GAINED
FOCUS_LOST
```

Following are the methods available in the FocusEvent class:

Component getOppositeComponent() – Returns the other component that gained or lost focus.

boolean isTemporary() – Method returns true or false that indicates whether the change in focus is temporary or not.

ItemEvent

An item event is generated when the user clicks on a check box or clicks a list item or selects / deselects a checkable menu item. These events are identified by the following constants:

SELECTED
DESELECTED

Following are the methods available in the ItemEvent class:

Object getItem() – Returns a reference to the item whose state has changed.

ItemSelectable getItemSelectable() – Returns the reference of ItemSelectable object that raised the event.

int getStateChange() – Returns the status of the state (whether SELECTED or DESELECTED)

KeyEvent

A key event is generated when a key on the keyboard is pressed, released or typed. These events are identified by the following constants:

KEY_PRESSED (when a key is pressed)
KEY_RELEASED (when a key is released)
KEY_TYPED (when a character is typed)

There are several other constants which recognizes various keys on the keyboard and they are as follows:

VK_ALT	VK_DOWN	VK_LEFT	VK_RIGHT
VK_CANCEL	VK_ENTER	VK_PAGE_DOWN	VK_SHIFT
VK_CONTROL	VK_ESCAPE	VK_PAGE_UP	VK_UP

KeyEvent is a sub class of InputEvent. Following are the methods available in KeyEvent class:

char getKeyChar() – Returns the character entered from the keyboard

int getKeyCode() – Returns the key code

MouseEvent

The mouse event is generated when the user generates a mouse event on a source. Mouse events are represented by the following constants:

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.

Following are some of the methods available in MouseEvent class:

int getX() – Returns the x-coordinate of the mouse at which the event was generated.

int getY() – Returns the y-coordinate of the mouse at which the event was generated.

Point getPoint() – Returns the x and y coordinates of mouse at which the event was generated.

int getClickCount() – This method returns the number of mouse clicks for an event.

int getButton() – Returns an integer that represents the buttons that caused the event. Returned value can be either NOBUTTON, BUTTON1 (left mouse button), BUTTON2 (middle mouse button), or BUTTON3 (right mouse button).

Following methods are available to obtain the coordinates relative to the screen:

Point getLocationOnScreen()

int getXOnScreen()

int getYOnScreen()

TextEvent

The text event is generated when the user enters a character into a text field or a text area. This event is identified by the following constant:

TEXT_VALUE_CHANGED

WindowEvent

A window event is generated when a user performs any one of the window related events which are identified by the following constants:

WINDOW_ACTIVATED	The window is activated
WINDOW_DEACTIVATED	The window was deactivated
WINDOW_OPENED	The window was opened
WINDOW_CLOSING	User requested the window to be closed
WINDOW_ICONIFIED	The window was iconified
WINDOW_DEICONIFIED	The window was deiconified
WINDOW_GAINED_FOCUS	The window gained input focus
WINDOW_LOST_FOCUS	The window lost input focus
WINDOW_CLOSED	The window is closed
WINDOW_STATE_CHANGED	The state of the window is changed

Following are some of the methods available in the WindowEvent class:

Window getWindow() – Returns the reference of the window on which the event is generated.

Window getOppositeWindow() – Returns the reference to the previous window when a focus event of activation event has occurred.

int getOldState() – Returns an integer indicating the old state of the window.

int getNewState() – Returns an integer indicating the new state of the window.

Sources of Events

All GUI elements which are derived from the Component class are examples of sources which generates events. Following are some of the examples of event sources available in Java:

Event Source	Description
Button	Generates action events when the button is pressed.
Check box	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.
Scroll bar	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.

Event Listener Interfaces

The event delegation model contains two main components. First are the event sources and second are the listeners. Most of the listener interfaces are available in the java.awt.event package. In Java, there are several event listener interfaces which are listed below:

ActionListener

This interface deals with the action events. Following is the event handling method available in the ActionListener interface:

void actionPerformed(ActionEvent ae)

AdjustmentListener

This interface deals with the adjustment event generated by the scroll bar. Following is the event handling method available in the AdjustmentListener interface:

void adjustmentValueChanged(AdjustmentEvent ae)

ComponentListener

This interface deals with the component events. Following are the event handling methods available in the ComponentListener interface:

void componentResized(ComponentEvent ce)

void componentMoved(ComponentEvent ce)

void componentShown(ComponentEvent ce)

void componentHidden(ComponentEvent ce)

ContainerListener

This interface deals with the events that can be generated on containers. Following are the event handling methods available in the ContainerListener interface:

void componentAdded(ContainerEvent ce)

void componentRemoved(ContainerEvent ce)

FocusListener

This interface deals with focus events that can be generated on different components or containers. Following are the event handling methods available in the FocusListener interface:

void focusGained(FocusEvent fe)

void focusLost(FocusEvent fe)

ItemListener

This interface deals with the item event. Following is the event handling method available in the ItemListener interface:

void itemStateChanged(ItemEvent ie)

KeyListener

This interface deals with the key events. Following are the event handling methods available in the KeyListener interface:

void keyPressed(KeyEvent ke)

void keyReleased(KeyEvent ke)

void keyTyped(KeyEvent ke)

MouseListener

This interface deals with five of the mouse events. Following are the event handling methods available in the MouseListener interface:

void mouseClicked(MouseEvent me)

void mousePressed(MouseEvent me)

void mouseReleased(MouseEvent me)

void mouseEntered(MouseEvent me)

void mouseExited(MouseEvent me)

MouseMotionListener

This interface deals with two of the mouse events. Following are the event handling methods available in the MouseMotionListener interface:

void mouseMoved(MouseEvent me)

void mouseDragged(MouseEvent me)

MouseWheelListener

This interface deals with the mouse wheel event. Following is the event handling method available in the MouseWheelListener interface:

void mouseWheelMoved(MouseWheelEvent mwe)

TextListener

This interface deals with the text events. Following is the event handling method available in the TextListener interface:

void textValueChanged(TextEvent te)

WindowFocusListener

This interface deals with the window focus events. Following are the event handling methods available in the WindowFocusListener interface:

void windowGainedFocus(WindowEvent we)

void windowLostFocus(WindowEvent we)

WindowListener

This interface deals with seven of the window events. Following are the event handling methods available in the WindowListener interface:

void windowActivated(WindowEvent we)

void windowDeactivated(WindowEvent we)

void windowIconified(WindowEvent we)

void windowDeiconified(WindowEvent we)

void windowOpened(WindowEvent we)

void windowClosed(WindowEvent we)

void windowClosing(WindowEvent we)

Using Delegation Event Model

Following are the basic steps in using delegation event model or for handling events in a Java program:

1. Implement the appropriate listener interface.
2. Register the listener with the source.
3. Provide appropriate event handler to handle the event raised on the source.

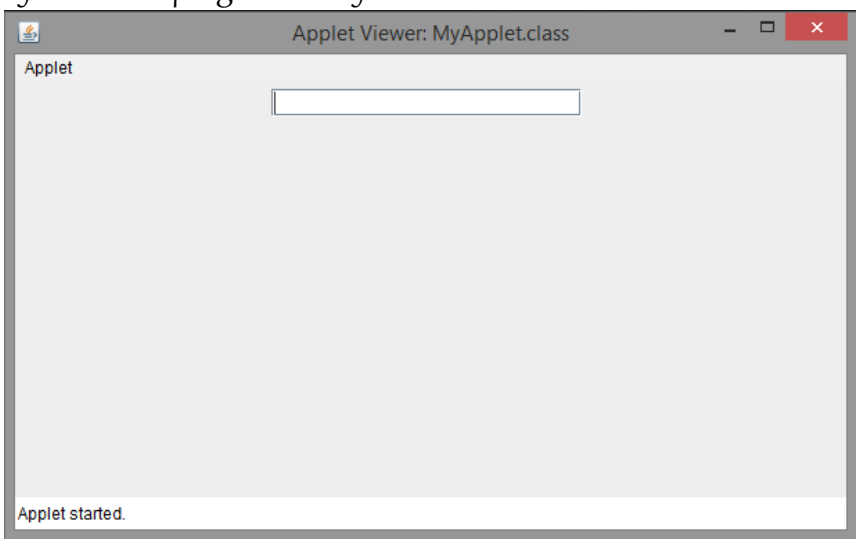
Key Events Handling

Following is a Java program which handles key events. In the program when the user types characters in the text field, they are displayed in a label below the text field:

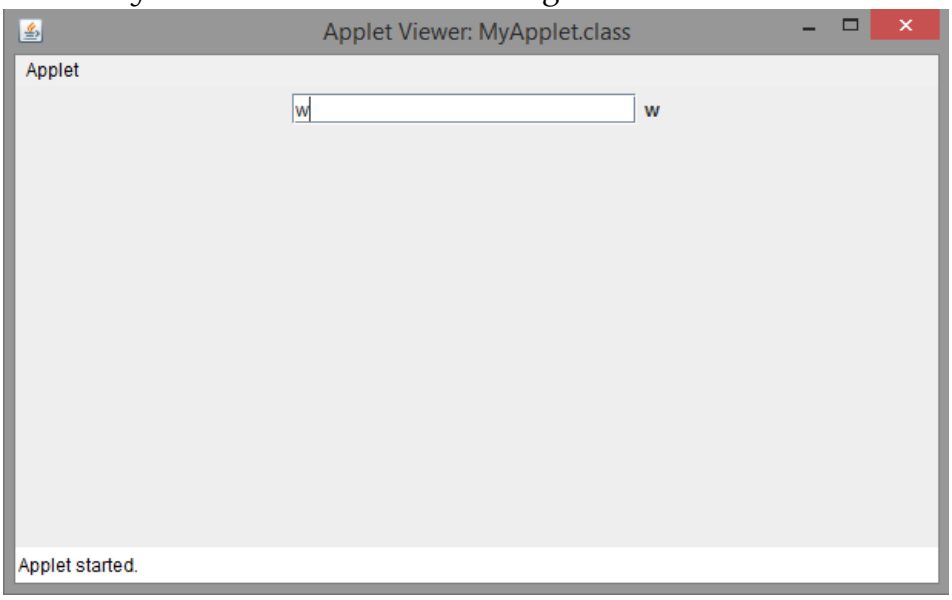
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class MyApplet extends JApplet implements KeyListener
{
        JTextField jtf;
        JLabel label;
        public void init()
```

```
    {
        setSize(600,300);
        setLayout(new FlowLayout());
        jtf = new JTextField(20);
        add(jtf);
        jtf.addKeyListener(this);
        label = new JLabel();
        add(label);
    }
    public void keyPressed(KeyEvent ke){}
    public void keyReleased(KeyEvent ke){}
    public void keyTyped(KeyEvent ke)
    {
        label.setText(String.valueOf(ke.getKeyChar()));
    }
}
```

Initial output of the above program is as follows:



After the user enters a character into the text field, the same character is displayed in the label beside the text field as shown in the below image:



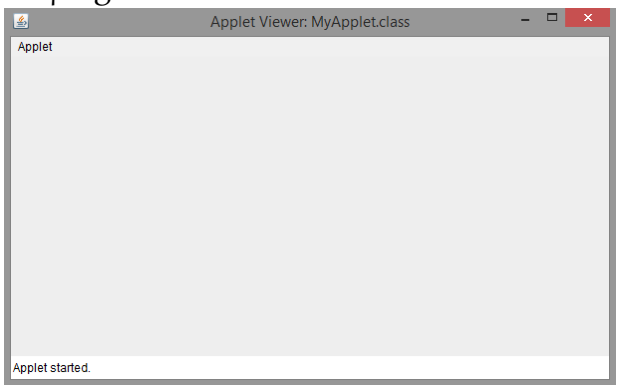
Mouse Events Handling

Following is a Java program which handles both mouse events and mouse motion events. When the user performs a mouse event on the applet it is updated in the label:

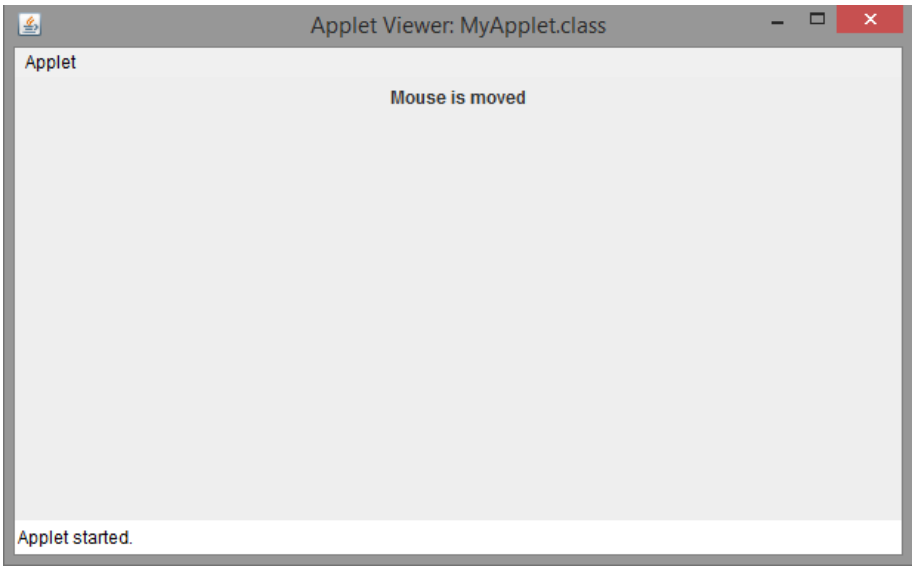
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class MyApplet extends JApplet implements MouseListener, MouseMotionListener
{
        JLabel label;
        public void init()
        {
                setSize(600,300);
                setLayout(new FlowLayout());
```

```
        label = new JLabel();
        add(label);
        addMouseListener(this);
        addMouseMotionListener(this);
    }
    public void mouseClicked(MouseEvent me)
    {
        label.setText("Mouse is clicked");
    }
    public void mousePressed(MouseEvent me){}
    public void mouseReleased(MouseEvent me){}
    public void mouseEntered(MouseEvent me){}
    public void mouseExited(MouseEvent me){}
    public void mouseDragged(MouseEvent me){}
    public void mouseMoved(MouseEvent me)
    {
        label.setText("Mouse is moved");
    }
}
```

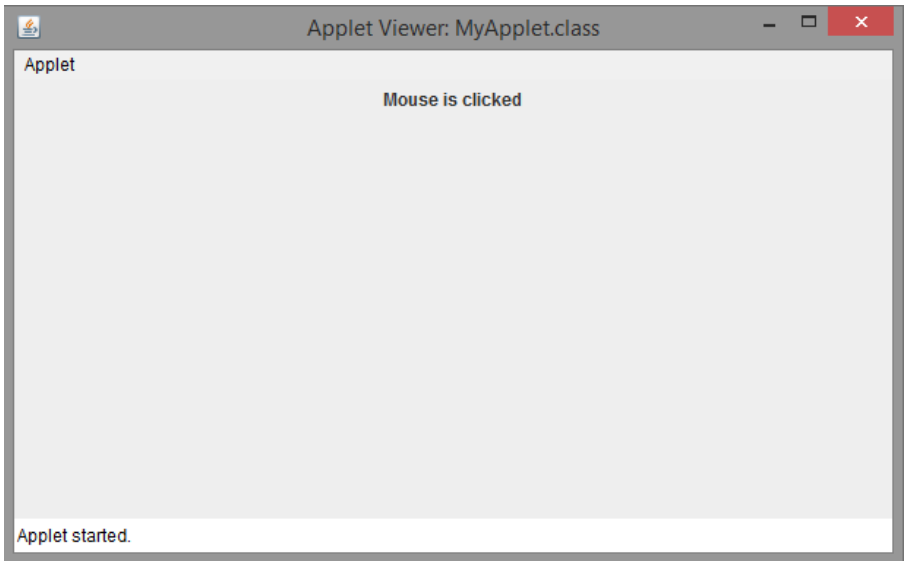
Initial output of the above program is as shown below:



When the user moves the mouse cursor over the applet display area, then the output is as shown below:



When the user clicks on the applet display area, the output is as shown below:



Adapter Classes

Adapter classes are a set of classes which can be used to simplify the event handling process. As we can see in the mouse event handling program here, even though we want to handle only one or two mouse events, we have to define all other remaining methods available in the listener interface(s).

To remove the aforementioned disadvantage, we can use adapter classes and define only the required event handling method(s). Some of the adapter classes available in the java.awt.event package are listed below:

Adapter Class	Listener Interface
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener and (as of JDK 6) MouseMotionListener and MouseWheelListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener, WindowFocusListener, and WindowStateListener

So, the mouse event handling program can be simplified using adapter classes as follows:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class MyApplet extends JApplet
{
        JLabel label;
        public void init()
```

```
	{
		setSize(600,300);
		setLayout(new FlowLayout());
		label = new JLabel();
		add(label);
		addMouseListener(new MyAdapter(label));
	}

}
class MyAdapter extends MouseAdapter
{
	JLabel label;
	MyAdapter(JLabel label)
	{
		this.label = label;
	}
	public void mouseClicked(MouseEvent me)
	{
		label.setText("Mouse is clicked");
	}
}
```

Inner Classes

An inner class is a class which is defined inside another class. The inner class can access all the members of an outer class, but vice-versa is not true. The mouse event handling program which was simplified using adapter classes can further be simplified using inner classes.

Following is a Java program which handles the mouse click event using inner class and adapter class:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class MyApplet extends JApplet
{
        JLabel label;
        public void init()
        {
                setSize(600,300);
                setLayout(new FlowLayout());
                label = new JLabel();
                add(label);
                addMouseListener(new MyAdapter());
        }
        class MyAdapter extends MouseAdapter
```

```
    {
        public void mouseClicked(MouseEvent me)
        {
            label.setText("Mouse is clicked");
        }
    }
}
```

In the above example, the class MyAdapter is the inner class which has been declared inside the outer class MyApplet.

Anonymous Inner Classes

Anonymous inner classes are nameless inner classes which can be declared inside another class or as an expression inside a method definition.

The above mouse event handling program can further be simplified by using an anonymous inner class as shown below:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class MyApplet extends JApplet
{
    JLabel label;
    public void init()
    {
```

```
            setSize(600,300);
            setLayout(new FlowLayout());
            label = new JLabel();
            add(label);
            addMouseListener(new MouseAdapter()
            {
                  public void mouseClicked(MouseEvent me)
                  {
                        label.setText("Mouse is clicked");
                  }
            }
            );
      }
}
```

In the above program, the syntax new MouseAdapter() {...} says to the compiler that the code written in between the braces represents an anonymous inner class and it extends the class MouseAdapter.

Whenever the line addMouseListener... executes, Java run-time system automatically creates the instance of anonymous inner class.