5.0 Introduction

Java applets are one of three kinds of Java programs:

- An *application* is a standalone program that can be invoked from the command line.
- An *applet* is a program that runs in the context of a browser session.
- A *servlet* is a program that is invoked on demand on a server program and that runs in the context of a web server process.

5.1 What is an Applet?

Applet is a small java program primarily used for Internet computing or *Applet* is a java program that can be embedded into a web page. They can run using ***Applet Viewer*** or any ***web browser*** that supports java.

Applet is embedded in a HTML page using the APPLET or OBJECT tag and hosted on a web server. Applets are used to make the web site more dynamic and entertaining. An applet can perform arithmetic operation, display graphics, play sounds, accept user input, create animation and play interactive games.

All applets are sub-classes (either directly or indirectly) of *java.applet.Applet* class. Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer which is a standard applet viewer tool provided by Java.

In general, execution of an applet does not begin at main() method. Output of an applet window is not performed by *System.out.println() r*ather it is handled with various AWT methods, such as *drawString().*

Advantages

- ✓ It works at client side so less response time.
- ✓ Applets are safe and secure.
- ✓ It can be executed by any java-enabled browser running under different platforms, including Linux, Windows, Mac OS etc.
- ✓ Automatically integrated with HTML; hence, resolved virtually all installation issues.
- ✓ Can provide dynamic, graphics capabilities and visualizations.
- ✓ Alternative to HTML GUI design.
- ✓ Can be launched as a standalone web application independent of the host web server

Disadvantages

- ✓ *Plug-in* is required at client browser to execute applet.
- ✓ Stability depends on stability of the client's web server
- ✓ Performance directly depend on client's machine
- ✓ Applets cannot read from or write to the files on the local computer.
- ✓ Applet cannot communicate with the other servers on the network.
- ✓ Applet cannot run any program from the local computer.
- ✓ Applet are restricted from using libraries from other languages such as C or C++. (Java language supports this feature through *native methods*)

Applet Types

Applets are of two types.

1. One which use the AWT (abstract window toolkit) to draw and display graphics.(Use Applet class)
2. Another type which uses Swing for drawing and display of graphics. (Use JApplet class).

The swing offers a richer and easier to use user interface than AWT. Swing based applets are more popular than AWT based. But AWT based applets are still used, especially when only a very simple user interface is required.

The *JApplet* class inherits the Applet class, so all the methods of Applet class are available in the Japplet class.

5.2 Applets v/s Applications

Features	Applet	Application
Definition	*Applet* is a small program primarily used for Internet computing	An application is a program executed on the computer independently.
main() method	Do not use the main method	Uses the main method for execution
Execution	Can not run independently require API's (Ex. Web API).	Can run alone but require JRE.
Nature	Requires some third party tool help like a browser to execute	Called as stand-alone application as application can be executed from command prompt
Read and write operation	Applets cannot read from and write to the files on the local computer.	Applications are capable of performing those operations to the files on the local computer.
Communication with other servers	Can not communicate with other servers.	Communication with other servers is probably possible.
Restrictions	Applets cannot access files residing on the local computer.	Can access any data or file available on the system.
Security	Requires security for the system as they are untrusted.	Does not require any security

5.4 Applet Life Cycle

Different states that an applet experiences between its object creation and object removal is known as *Applet Life Cycle*. There exists 4 states represented by 5 methods. These methods are known as *"callback methods"* as they are called automatically by the browser whenever required for the smooth execution of the applet.

Following are the methods.

1. init() method
2. start() method
3. paint() method
4. stop() method
5. destroy() method

An Applet is a subclass of *java.applet* package. These methods are defined in *Applet* class except paint() method. The *paint()* method is defined in java.awt.Component class, an indirect super class of Applet.

Browser Responsibilities

The Applet Life Cycle methods are called as *callback methods* as they are called implicitly by the browser for the smooth execution of the applet. Browser should provide an environment known as container for the execution of the applet.

Following are the responsibilities of the browser.

1. It should call the callback methods at appropriate times for the smooth execution of the applet.
2. It is responsible to maintain the Applet Life Cycle.
3. It should have the capability to communicate between applets, applet to JavaScript and HTML, applet to browser etc.

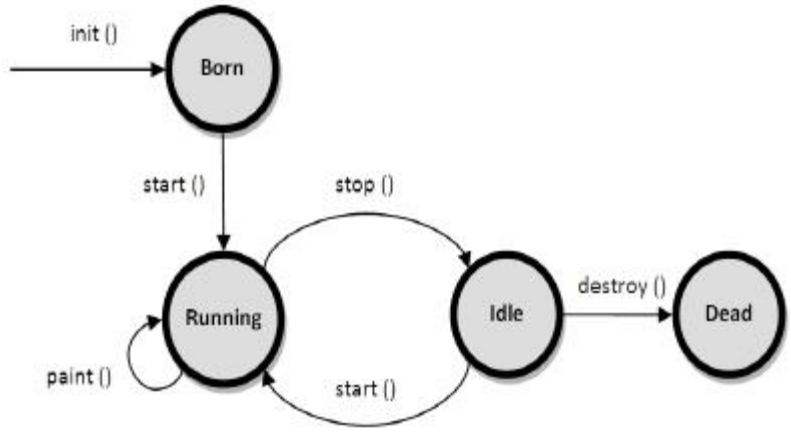


Fig. Applet Life Cycle

Description of Life Cycle Methods

1. public void init()

This method is called at the time of starting the execution. This is called only once in the life cycle. This method initializes the Applet and it helps to initialize variables and instantiate the objects and load the GUI of the applet. This is invoked when the page containing the applet is loaded and places the applet in *new born* state. Its gereral form is:

```
public void init()
{
        statements;
}
```

2. start() method

The start() method executes immediately after the *init()* method. It starts the execution of Applet. In this state, the applet becomes active. The *start()* method contains the actual code of the applet that should run. It also executes whenever the applet is restored, maximized or moving from one tab to another tab in the browser.

```
public void start()
{
        statements;
}
```

3. stop() method

The *stop()* method stops the execution of the applet. It is invoked when the Applet or the browser is minimized. The Applet frequently comes to this state in its life cycle and can go back to its active state.

```
public void stop()
{
        statements;
}
```

4. paint() method

This method helps to create Applet's GUI such as a colored background, drawing and writing. The *paint()* method executes after the execution of *start()* method and whenever the applet or browser is resized.

```
public void paint(Graphics g)
{
        statements;
}
```

This method takes a *java.awt.Graphics* object as parameter.

5. destroy() method

This destroys the Applet and is also invoked only once when the active browser page containing the applet is closed.

```
public void destroy( )
{
        statements;
}
```

Order of Execution

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()

5.5 Structure/Skeleton of Applet

```
import java.awt.*;
import java.applet.*;
/*
        <applet code="MyApplet" width=pixels height=pixels>
        </applet>
*/
public class MyApplet extends Applet
{
        public void init() {
                // initialization
        }
        public void start() {
                // start or resume execution
        }
        public void stop() {
                // suspends execution
        }
        public void destroy() {
                // perform shutdown activities
        }
        public void paint(Graphics g) {
                // redisplay contents of window
```

```
        }
}
```

How to run an Applet?

There are two ways to run an applet

1. By appletViewer tool (for testing purpose).
2. By html file. (Save the code with .html extesion)

Filename.html

```
<html>
        <applet code = “filename.class" width = "xxx" height = "xxx">
        </applet>
</html>
```

5.6 Java API support for Applets

Java includes many libraries *(a.k.a packages)* and classes, commonly known as *Java API*. *java.applet* is a small package used to develop applets. It contains 3 interfaces and 1 class. They are:

Interfaces

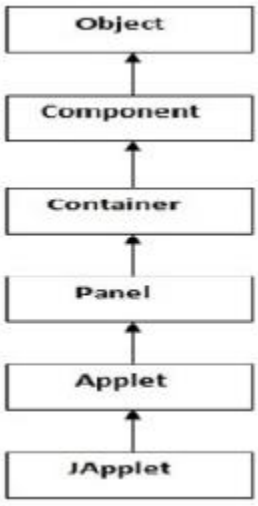
1. *AppletContext:* Used by the programmer to communicate with the execution environment of an applet. This is required to communicate between two applets.
2. *AppletStub:* Used to develop custom applet viewers. Programmer can utilize this interface to communicate between applet and the browser environment.
3. *AudioClip:* Used to play audio clips. It comes with methods like *play(), loop() and stop()*.

Classes

1. *Apple:* This class should be extended by our applet program. Applet is a class inherited from Panel of *java.awt* package.

Applet class provides all necessary support for applet execution, such as initializing and destroying of applet. It also provides methods that load and display images and methods that load and play audio clips.

Hierarchy of Applet class



5.7 Creating and executing Applet

1. Create an html file with <Applet > tag (MyApplet.html)
2. Create a java code for applet (HelloApplet.java)
3. Compile code and get class file (HelloApplet.class)
4. Run the html file(By web browser or by appletviewer)

Step1: Create an html file with <Applet > tag

MyApplet.html

```
<html>
        <body>
                <applet code= "MyApplet.class" width="200" height="200">
                </applet>
        </body>
</html>
```

Step2: Create a java code for applet

MyApplet.java

```
import java.applet.*;
import java.awt.Graphics;
public class MyApplet extends Applet
{
        public void paint(Graphics g)
        {
                g.drawString("welcome",150,150);
        }
 }
```

Step 3: Compile code and get class file

C:/> javac MyApplet.java

Step 4: Run the html file\

C:/> appletviewer MyApplet.html

paint( ) method

The *paint()* method is called each time your applet's output must be drawn/redrawn. For example, the window in which the applet is running may be overwritten by another window and then uncovered. Or the applet window may be minimized and then restored. The *paint( )*

method has one parameter of type *Graphics* class , which describes the graphics environment in which the applet is running.

Update() method

This method is called when your applet has requested that a portion of its window be redrawn. The default version of *update()* first fills an applet with the default background color and then calls *paint( )* method to paint the rest of the component.

```
public void update(Graphics g)
{
        Color bg = getBackground();
        Color fg = getForeground();
}
```

repaint() method

Calling a repaint () method causes the whole component to be repainted.

repaint() -> update() ->paint()

The repaint( ) method has four forms.

void repaint( )
void repaint(int *left*, int *top*, int *width*, int *height*)
void repaint(long *maxDelay*)
void repaint(long *maxDelay*, int *x*, int *y*, int *width*, int *height*)

5.8 Displaying Graphics in Applet

java.awt.Graphics class provides many methods for graphics programming.

Methods

Method	Description
public abstract void drawString(String str, int x, int y)	used to draw the specified string.
public void drawRect(int x, int y, int width, int height)	Draw a rectangle with the specified width and height.
public abstract void fillRect(int x, int y, int width, int height)	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)	Used to draw oval with the specified width and height.
public abstract void fillOval(int x, int y, int width, int height)	Used to fill oval with the default color and specified width and height.
public abstract void drawLine(int x1, int y1, int x2, int y2)	Used to draw line between the points(x1, y1) and (x2, y2).
public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer)	Used to draw the specified image.
public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)	Used to draw a circular or elliptical arc.
public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)	Used to fill a circular or elliptical arc.
public abstract void setColor(Color c)	Used to set the graphics current color to the specified color.
public abstract void setFont(Font font)	Used to set the graphics current font to the specified fonts

5.9 The applet tag

Applets are embedded in HTML documents with the *<APPLET> tag*. It contains attributes that identify the applet to be displayed and, optionally, give the web browser hints about how it should be displayed.

Syntax:

```
<APPLET
        [CODEBASE = CODEBASE_URL]
        CODE=class name
        [ALT=alternate_text]
        [NAME = applet_instance_name]
        WIDTH = pixels HEIGHT= pixels
        [ALIGN = alignment]
        [VSPACE =pixels]
        [ HSPACE =pixels]
>
        [ <PARAM NAME =name1 VALUE=value1>]
        [ <PARAM NAME =name2 VALUE=value2>]
        …….
        [Text to be displayed in the absence of java]
</APPLET>
```

Attributes

1. *CODEBASE:* This attribute is used to specify the base URL of the applet -- the directory or folder that contains the applet's code.
2. *CODE:* This required attribute gives the name of the file that contains the applet's compiled Applet subclass.
3. *ALT:* Any text that should be displayed is specified by this optional attribute if the browser understands the APPLET tag but can't run Java applets.
4. *NAME***:** This optional attribute specifies a name for the applet instance. With the help of this attribute it is possible for applets on the same page to find (and communicate with) each other.
5. *WIDTH, HEIGHT:* The initial width and height of the applet is given by these attributes.
6. *ALIGN***:** This attribute specifies the alignment of the applet with the following possible values: *left, right, top, texttop, middle, absmiddle, baseline, bottom, absbottom*
7. *VSPACE, HSPACE:* These optional attributes are used to specify the number of pixels above and below the applet (VSPACE) and on each side of the applet (HSPACE).
8. *<PARAM>:* The only way to specify applet-specific parameters is to use the <PARAM> tags. Applets read user-specified values for parameters with the **getParameter()** method.

5.10 Passing parameters to an applet

Java applet has the feature of retrieving the parameter values passed from the HTML page. So, you can pass the parameters from your html page to the applet embedded in your page. Parameters specify extra information that can be passed to an applet from the HTML

page. Parameters are passed to applets in the form of (***name, value***) pairs using ***<PARAM>*** tag which is a sub tag of *<APPLET>* tag.

Inside the applet, you read the values passed through the ***<PARAM>*** tag with the ***getParameter()*** method of the Applet class.

The *getParameter()* method of the *Applet* class can be used to retrieve the parameters passed from the HTML page. Its general form is:

String getParameter(String param-name);

Example

Param.html

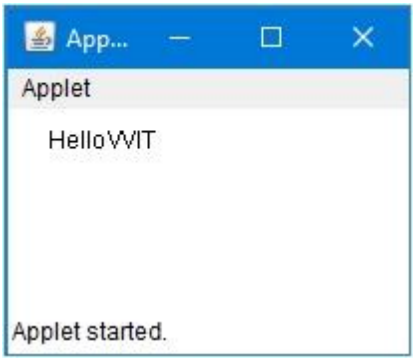
```
<html>
<body>
        <applet code="MyApplet" height="100" width="200">
                <param name="wish" value="Hello" />
                <param name="clg" value="VVIT" />
        </applet>
</body>
</html>
```

Myapplet.java

```
import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet
{
        String w, c;
        public void init()
        {
                w = getParameter("wish");
                c = getParameter("clg");
        }
        public void paint(Graphics g)
        {
                g.drawString(w, 20, 20);
                g.drawString(c, 50, 20);
        }
}
```

Output

C:/> javac MyApplet.java
C:/>appletviewer Param.html



5.11 Accepting input from user

Applet works in graphical environment. So applet treats the input as text string. The TextField class of applet package is used for creating a text field on applet. The input from text field is in string form, so it should be changed to right format before any computation. For displaying the output the result should be converted to String.

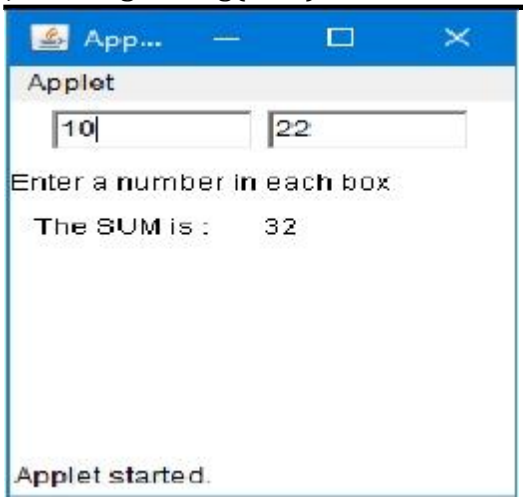
Example

```
import java.awt.*;
import java.applet.*;
public class UserInput extends Applet
{
	TextField text1,text2;
	public void init()
	{
		text1 = new TextField(8);
		text2 = new TextField(8);
		add(text1);
		add(text2);
		text1.setText("0");
		text2.setText("0");
	}
	public void paint(Graphics g)
	{
		int x=0,y=0,z=0;
		String s1,s2,s;
		g.drawString("Enter a number in each box ",0,50);
		try
		{
			s1=text1.getText();
			x=Integer.parseInt(s1);
			s2=text2.getText();
			y=Integer.parseInt(s2);
		}
		catch(Exception e){}
		z=x+y;
		s=String.valueOf(z);
		g.drawString("The SUM is :",10,75);
		g.drawString(s,100,75);
	}
	public boolean action(Event event, Object obj)
	{
		repaint();
		return true;
	}
}
/*
	<applet code="UserInput.class" height="200" width="200">
	</applet>
*/
```

Output:

C:/> javac UserInput.java

C:/>appletviewer UserInput.java



Event Handling

Event handling is at the core of successful applet programming. Events are supported by the *java.awt.event* package. These events are passed to your applet in a variety of ways, With the specific method depending upon the actual event.

There are several types of events. The most commonly handled events are those generated by the mouse, the keyboard, and various controls, such as a push button.

5.12 Delegation Event model

The *Delegation Event model* is one of the sophisticated techniques used to handle events in GUI programming languages. The modern approach to handling events is based on the delegation event model, which defines standard and consistent mechanisms to generate and process events.

The delegation event model provides a standard mechanism for a source to generate an event and send it to a set of listeners. The listener simply waits until it receives an event. Once received, the listener processes the event and then returns.

The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events. A user interface element is able to *"delegate"* the processing of an event to a separate piece of code.

In the delegation event model, listeners must register with a source in order to receive an event notification. This provides an important benefit: notifications are sent only to listeners that want to receive them. There are mainly three parts in delegation event model.

1. Events.
2. Event sources.
3. Event Listeners.

Events

An *event* is an object that describes a state change in a source. It can be generated as a consequence of a person interacting with the elements in a graphical user interface. Some of the activities that cause events to be generated are *pressing a button*, *entering a character* via the keyboard, *selecting an item in a list* and *clicking the mouse.*

Events may also occur that are not directly caused by interactions with a user interface. For example, an event may be generated when a timer expires, a counter exceeds a value, software or hardware failure occurs, or an operation is completed.

Event Sources

A *source* is an object that generates an *event*. This occurs when the internal state of that object changes in some way. Sources may generate more than one type of event. A *source* must register listeners in order for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method. Here is the general form:

```
public void addTypeListener(TypeListener 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()*.

When an *event* occurs, all registered listeners are notified and receive a copy of the event object. This is known as *multicasting* the event. In all cases, notifications are sent only to listeners that register to receive them

Some sources may allow only one listener to register. The general form of such a method is this:

```
public void addTypeListener(TypeListener el) throws java.util.TooManyListenersException
```

Here, *Type* is the name of the event and el is a reference to the event listener. When such an event occurs, the registered listener is notified. This is known as *unicasting* the event.

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 removeTypeListener(TypeListener 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(). The *methods* that *add or remove listeners* are provided by the source that generates events. For example, the Component class provides methods to add and remove keyboard and mouse event listeners.

Event Listeners

A ***listener*** is an object that is notified when an event occurs. It has two major requirements.

1. First, it must have been registered with one or more sources to receive notifications about specific types of events.
2. 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*. For example, the *MouseMotionListener* interface defines two methods to receive notifications when the mouse is dragged or moved. Any object may receive and process one or both of these events if it provides an implementation of this interface.

5.13 Event classes

The classes that represent events are at the core of Java's event handling mechanism. They provide a consistent, easy-to-use means of encapsulating events. At the root of the Java event class hierarchy is *EventObject*, which is in *java.util*. It is the superclass for all events.

The class *AWTEvent* defined within the *java.awt* package, is a subclass of *EventObject*. It is the superclass for all AWT-based events used by the delegation event model.

The package *java.awt.event* defines several types of events that are generated by various user interface elements.

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 or moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
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: The Main Event Classes in **java.awt.event**

5.14 Event Listener Interfaces

Event listeners receive event notifications. Listeners are created by implementing one or more of the interfaces defined by the *java.awt.event* package. When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its

argument.

Interface	Description
ActionListener	Defines one method to receive action events. Action events are generated by such things as push buttons and menus.
AdjustmentListener	Defines one method to receive adjustment events, such as those produced by a scroll bar.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes. An item event is generated by a check box, for example.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
TextListener	Defines one method to recognize when a text value changes.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Table: Common Event Listener Interfaces

Event Listener Interfaces and their methods

Interface	Interface Methods	Add Method	Components
WindowListener	windowActivated(WindowEvent) windowClosed (WindowEvent) windowClosing (WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified (WindowEv	addWindiwListener()	Frame ,
ActionListener	actionPerformed (ActionEvent)	addActionListener()	Button, List, MenuItem, TextField
AdjustmentEvent	adjustmentValueChanged (AdjustmentEvent)	addAdjustmentListener( )	Scrollbar
ComponentListener	componentMoved (ComponentEvent) componentHidden (ComponentEvent) componentResized (ComponentEvent) componentShown(ComponentEvent)	addComponentListener( )	Canvas
ContainerListener	componentAdded (ContainerEvent) componentRemoved (ContainerEvent)	addContainerListener( )	Frame , Panel
FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)	addFocusListener( )	Frame , Button, Canvas
ItemListener	itemStateChanged (ItemEvent)	addItemListener( )	Checkbox, Choice, List
KeyListener	keyPressed (KeyEvent) keyReleased (KeyEvent)	addKeyListener( )	Button , Canvas, Panel

	keyTyped (KeyEvent)		
MouseListener	mouseClicked (MouseEvent) mouseEntered (MouseEvent) mouseExited (MouseEvent) mousePressed (MouseEvent) mouseReleased (MouseEvent)	addMouseListener( )	Button , Canvas, Panel
MouseMotionListener	mouseDragged (MouseEvent) mouseMoved (MouseEvent)	addMouseMotionListener( )	Button , Canvas, Panel
TextListener	textValueChanged (TextEvent)	addTextListener( )	TextComponent

5.15 Handling Mouse Events

To handle mouse events, you must implement the **MouseListener** and the **MouseMotionListener** interfaces.

The **MouseListener** interface defines five methods. If a mouse button is clicked, **mouseClicked( )** is invoked. When the mouse enters a component, the **mouseEntered( )** method is called. When it leaves, **mouseExited( )** is called. The **mousePressed( )** and **mouseReleased( )** methods are invoked when a mouse button is pressed and released, respectively. The general forms of these methods are shown here:

1. void mouseClicked(MouseEvent *me*)
2. void mouseEntered(MouseEvent *me*)
3. void mouseExited(MouseEvent *me*)
4. void mousePressed(MouseEvent *me*)
5. void mouseReleased(MouseEvent *me*)

The **MouseMotionListener** interface defines two methods. The **mouseDragged( )** method is called multiple times as the mouse is dragged. The **mouseMoved( )** method is also called multiple times as the mouse is moved. Their general forms are shown here:

1. void mouseDragged(MouseEvent *me*)
2. void mouseMoved(MouseEvent *me*)

The **MouseEvent** object passed in *me* describes the event. **MouseEvent** defines a number of methods but the most commonly used methods are **getX( )** and **getY( )**. These return the X and Y
coordinates of the mouse when the event occurred. Their forms are shown here:

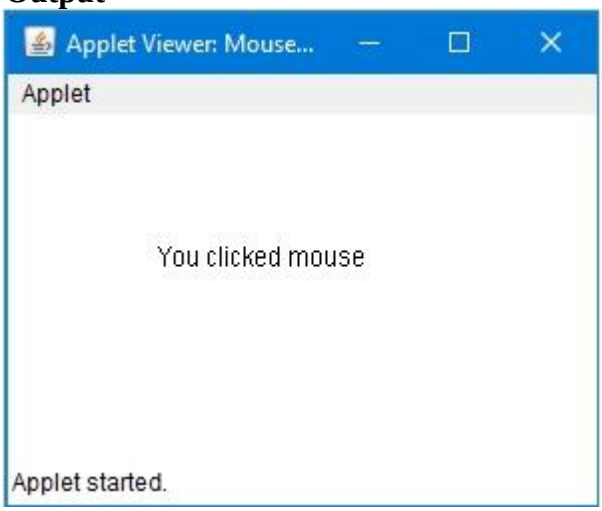
1. int getX( )
2. int getY( )

Example: Java program using listeners for handling mouse events

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/*
        <applet code="MouseEvents.class" height="300" width="300">
        </applet>
*/
public class MouseEvents extends Applet implements MouseListener,MouseMotionListener
{
        String str="";
        public void init()
        {
                addMouseListener(this);
                addMouseMotionListener(this);
```

```
        }
        public void mousePressed(MouseEvent me) {
                str ="You pressed mouse";
                repaint();
        }
        public void mouseReleased(MouseEvent me) {
                str ="You released mouse";
                repaint();
        }
        public void mouseClicked(MouseEvent me) {
                str ="You clicked mouse";
                repaint();
        }
        public void mouseEntered(MouseEvent me) {
                str ="You entered applet window";
                repaint();
        }
        public void mouseExited(MouseEvent me) {
                str ="You exited applet window";
                repaint();
        }
        public void mouseDragged(MouseEvent me) {
                str ="You are dragging";
                repaint();
        }
        public void mouseMoved(MouseEvent me) {
                str ="You are moving mouse";
                repaint();
        }
        public void paint(Graphics g)
        {
                g.drawString(str,75,75);
        }
}
```

Output



5.16 Handling Keyboard Events class

To handle keyboard events, you must implement the **KeyListener** interface. The **KeyListener** interface defines three methods. The **keyPressed( )** and **keyReleased( )**

methods are invoked when a key is pressed and released, respectively. If any alpha numeric key is pressed, **keyTyed()** is invoked. The general forms of these methods are shown here:

1. void keyTyped(KeyEvent k*e*)
2. void keyPressed(KeyEvent k*e*)
3. void keyReleased(KeyEvent k*e*)

KeyEvent class

An event which indicates that a keystroke occurred in a component. It defines several constants. They are:

1. **public static final int KEY_PRESSED:** The "key pressed" event. This event is generated when a key is pushed down.
2. **public static final int KEY_RELEASED:** The "key released" event. This event is generated when a key is let up.
3. **public static final int KEY_TYPED:** The "key typed" event. This event is generated when a character is entered.

There are many other integer constants that are defined by KeyEvent. For example:
VK_0 to VK_9, VK_A to VK_Z define the ASCII equivalents of the numbers and letters.

Here are some others:
VK_ENTER, VK_ESCAPE, VK_CANCEL, VK_UP, VK_DOWN, VK_LEFT, VK_RIGHT, VK_PAGE_DOWN,VK_PAGE_UP, VK_SHIFT, VK_ALT, VK_CONTROL

The VK constants specify virtual key codes and are independent of any modifiers, such as control, alt or shift,

Methods

1. *public int getKeyCode():* Returns the integer keyCode associated with the key in this event. Returns the integer code for an actual key on the keyboard.
2. *public char getKeyChar():* Returns the character associated with the key in this event. For example, the KEY_TYPED event for shift + "a" returns the value for "A".

Java program that illustrates Keyboard events

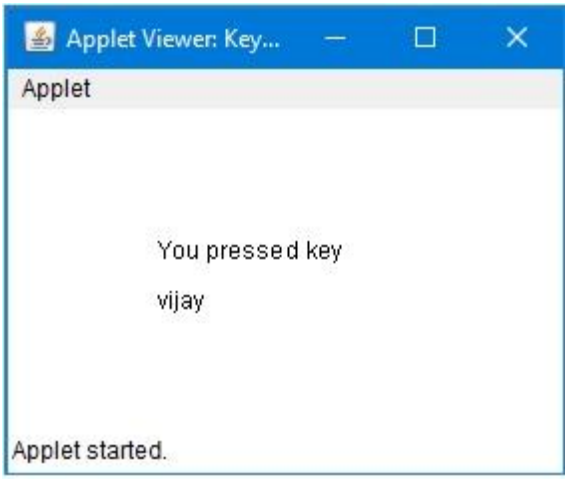
```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/*
        <applet code="KeyboardEvents.class" height="200" width="300">
        </applet>
*/
public class KeyboardEvents extends Applet implements KeyListener
{
        String str="",str1="";
        public void init() {
                addKeyListener(this);
        }
        public void paint(Graphics g) {
                g.drawString(str,75,75);
                g.drawString(str1,75,100);
        }
        public void keyPressed(KeyEvent ke) {
                str ="You pressed key";
```

```
                repaint();
        }
        public void keyReleased(KeyEvent ke) {
                str ="You released key";
                repaint();
        }
        public void keyTyped(KeyEvent ke) {

                str1 += ke.getKeyChar();
                repaint();
        }
}
```

Output

javac KeyboardEvents.java
appletviewer KeyboardEvents.java



5.17 What is adapter class?

An adapter class provides the default implementation of all methods in listener interface. Adapter classes are very useful when you want to process only few of the events that are handled by a particular event listener interface. 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*.

The adapter classes are found in

1. java.awt.event
2. java.awt.dnd and
3. javax.swing.event packages.

Java.awt.event Adapter classes

Adapter class	**Listener Interface**
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener

5.18 Inner classes

Java *inner class* or *nested class* is a class which is declared inside the class or interface. They enable you to logically group classes and interfaces in one place, thus increase the use of encapsulation. The scope of a nested class is bounded by the scope of its enclosing class. A *nested class* can access all the members of outer class including private data members and methods. However, reverse is not true.

A nested class is also a member of its enclosing class. A nested class can be declared private, public, protected, or package private (default).

Advantages:

1. Nested classes represent a special type of relationship that is it can access all the members of outer class including private.
2. Nested classes are used to develop more readable and maintainable code because it logically group classes and interfaces in one place only.
3. Code Optimization: It requires less code to write.

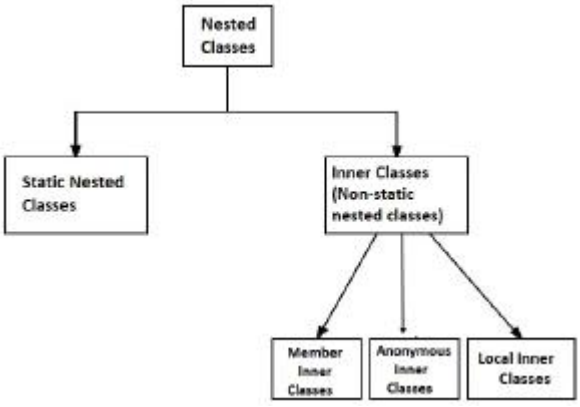
Syntax of Inner class

```
class Java_Outer_class
{
        //code
        class Java_Inner_class
        {
                //code
        }
  }
```

Types of Nested classes

There are two types of nested classes:

1. Non-static nested class (inner class)
    a. Member inner class
    b. Anonymous inner class
    c. Local inner class
2. Static nested class



Inner class is a part of nested class. Non-static nested classes are known as inner classes.

Description

Class	Description
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing interface or extending class. Its name is decided by the java compiler.
Local Inner Class	A class created within method.
Static Nested Class	A static class created within class.

Member inner class

A non-static class that is created inside a class but outside a method is called member inner class.

Java program that illustrates Member inner classes

```
class Outer {
        private int a=30;
        class Inner
        {
                void msg( ) {
                        System.out.println("a = "+a);
                }
        }
        public static void main(String args[])
        {
                Outer o=new Outer();
                Outer.Inner in=o.new Inner();
                in.msg();
        }
}
```

Java Anonymous inner class

A class that has no name is known as anonymous inner class in java for which only a single object is created. It should be used if you have to override method of an *class* or *interface*. Anonymous inner classes are useful in writing implementation classes for listener interfaces in graphics programming.

Java Anonymous inner class can be created by two ways:

1. Class (may be abstract or concrete).
2. Interface

Java program that illustrates Anonymous inner class

```
interface Age
{
    int x = 21;
    void getAge();
}
class MyClass {
    public static void main(String[] args)
    {
         Age obj = new Age()
        {
            public void getAge()
            {
                System.out.print("Age is "+x);
            }
        };
        obj.getAge();
    }
}
```

Output :

Age is 21

Local Inner class

A class i.e. created inside a method is called local inner class in java. If you want to invoke the methods of local inner class, you must instantiate this class inside the method.

Java program that illustrates local inner class

```
public class localInner
{
        private int a=10;        //instance variable
        void display()
        {
                class Local
                {
                        void msg( )
                        {
                                System.out.println(a);
                        }
                }
                 Local l=new Local();
                 l.msg();
        }
        public static void main(String args[])
        {
                 localInner li=new localInner1();
                 li.display();
        }
}
```

Output : 30

Static nested classes

A static class that is created inside a class is called *static nested class* in java. It cannot access non-static data members and methods. It can be accessed by outer class name.

1. It can access static data members of outer class including private.
2. Static nested class cannot access non-static (instance) data member or method.

Java program that illustrates static nested classes

```
class Outer
{
        static int s=10;
        static class Inner
        {
                void msg()
                {
                        System.out.println("s = "+s);
                }
        }
        public static void main(String args[])
        {
                Outer.Inner obj=new Outer.Inner();
                obj.msg();
        }
}
```

Output: s=100