Java String

Generally, string is a sequence of characters. But in java, string is an object that represents a sequence of characters. The java.lang.String class is used to create string object.

The Java platform provides the String class to create and manipulate strings.

simply a string is an array of characters.. For example:

```
    char[] ch={'H','e','l','l','o'};
String s=new String(ch);
```

Is same as:
String s="Hello";

Java String class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

There are two ways to create a Java string.

1. By string literal
2. By new keyword

1.Using a string literal in Java

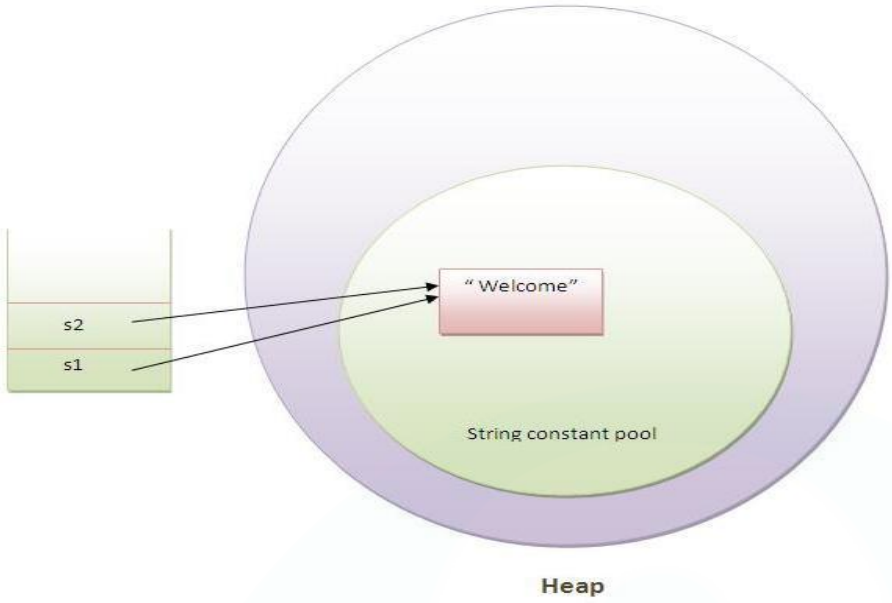
Java String literal is created by using double quotes. For Example:

String s="welcome";

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

```
String s1="Welcome";
String s2="Welcome"; //will not create new instance
```

In the above example only one object will be created. Firstly JVM will not find any string object with the value "Welcome" in string constant pool, so it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create new object but will return the reference to the same instance.



Java uses concept of string literal to make Java more memory efficient (because no new objects are created if it exists already in string constant pool).

2. Using a new Java keyword

String s = new String ("Welcome");

➢ In such case, JVM will create a new string object in normal(non pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in heap(non pool).

String Constructors

The **String** class supports several constructors as follows:

1. To create an empty **String**, we call the default constructor.

Example,

String s = new String(); will create an instance of **String** with no characters in it.

2. To create a **String** initialized by an array of characters, use the constructor shown here:

String(char chars[ ])

Example,

```
char chars[] = { 'a', 'b', 'c' };
String s = new String(chars);
```

This constructor initializes **s** with the string "abc".

3. We can specify a subrange of a character array as an initializer using the following constructor:

String(char chars[ ], int startIndex, int numChars)

Here, startIndex specifies the index at which the subrange begins, and numChars specifies the number of characters to use.

Example,

```
char chars[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
String s = new String(chars, 2, 3);
```

This initializes **s** with the characters **cde**.

4. We can construct a **String** object that contains the same character sequence as another **String** object using this constructor:

```
String(String strObj)    // Here, strObj is a String object.
```

Example,

```
char c[ ] = {'J', 'a', 'v', 'a'};
String s1 = new String (c);
String s2 = new String (s1);
```

Java String Example

```
public class StringExample
{
    public static void main(String args[])
    {
        String s1="java";//creating string by java string literal
        char ch[]={'s','t','r','i','n','g','s'};
        String s2=new String(ch);//converting char array to string
        String s3=new String("example");//creating java string by new keyword
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
    }
}
```

Output:

```
java
strings
example
```

String functions

charAt()

This function returns the character located at the specified index.

```
String str = "studytonight";
System.out.println(str.charAt(2));//u
```

substring()

This method returns a part of the string. substring() method has two forms,

public String substring(int begin);// This method returns the ith index substring.

public String substring(int begin, int end); // This method in java returns the substring from i to j-1 index. Character of begin index is inclusive and character of end index is exclusive.

```
String str = "0123456789";
System.out.println(str.substring(4));//456789
System.out.println(str.substring(4,7));//456
```

concat()

This method concatenates the specified string to the end of this string.

```
String s1="Sachin ";
String s2="Tendulkar";
String s3=s1.concat(s2);
System.out.println(s3); //Sachin Tendulkar
String s="Sachin"+" Tendulkar";
System.out.println(s);//Sachin Tendulkar
```

indexOf ()

This method returns the index of first occurrence of a substring or a character. indexOf()method has four forms:

1. int indexOf(String str)
2. int indexOf(int ch, int fromIndex)
3. int indexOf(int ch)
4. int indexOf(String str, int fromIndex)

```
String str="StudyTonight";

System.out.println(str.indexOf('u'));   //2 (3rd form)

System.out.println(str.indexOf('t', 3));    //11(2nd form)

String subString="Ton";

System.out.println(str.indexOf(subString)); //5(1st form)

System.out.println(str.indexOf(subString,7));
```

//-1 (4th form). -1 indicates that the substring/ Character is not found in the given String.

equals() and equalsIgnoreCase()

The String equals() method compares the original content of the string. It compares values of string for equality. String class provides two methods:

- **public boolean equals(Object another)** compares this string to the specified object.
- **public boolean equalsIgnoreCase(String another)** compares this String to another string, ignoring case.

```
String s1="Sachin";
String s2="SACHIN";
System.out.println(s1.equals(s2));//false
System.out.println(s1.equalsIgnoreCase(s2));//true
```

compareTo()

The String compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:
s1 == s2 : 0
s1 > s2 : positive value
s1 < s2 : negative value

```
String s1="Sachin";
String s2="Sachin";
String s3="Ratan";
System.out.println(s1.compareTo(s2));//0
System.out.println(s1.compareTo(s3));//1(because s1>s3)
System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
```

String toUpperCase() and toLowerCase()

The java string toUpperCase() method converts this string into uppercase letter and string toLowerCase() method into lowercase letter.

```
String s="Sachin";
System.out.println(s.toUpperCase());//SACHIN
System.out.println(s.toLowerCase());//sachin
```

trim()

This method eliminates white spaces at both ends. It does not affect whitespaces in the middle.

```
String s="  Sachin  ";
System.out.println(s);//  Sachin
System.out.println(s.trim());//Sachin
```

replace(char old, char new)

This method returns new string by replacing all occurrences of oldChar with newChar.

```
String s1="java";
String replaceString=s1.replace('a','e');//jeve , replaces all occurrences of 'a' to 'e'
```

length()

The string length() method returns length of the string.

```
String s="Sachin";
System.out.println(s.length());//6
```

startsWith() and endsWith() method

```
String s="Sachin";
System.out.println(s.startsWith("Sa"));//true
System.out.println(s.endsWith("n"));//true
```

charAt()

The string charAt() method returns a character at specified index.

```
String s="Sachin";
System.out.println(s.charAt(0));//S
System.out.println(s.charAt(3));//h
```

Java Applet

Applet is a Java program that can be embedded into a web page. It runs inside the web browser and works at client side. 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.

Features of Java Applets

The Applet class provides the standard interface between the applet and the browser environment.

All applets are subclasses of **Applet** class. Thus, all applets must import **java.applet**. Applet

Applets must also import **java.awt**.

Applet do not use the main() method for initiating the execution of the code. Applets, when loaded automatically call certain methods of applet class to start and execute the applet code.

Unlike standalone applications, applet cannot be run independently. They are run from inside a webpage using a special feature known as **HTML** tag.

Applet cannot communicate with other servers on the network.

Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer. JDK provides a standard applet viewer tool called applet viewer.

Output of an applet window is not performed by System.out.println(). Rather it is handled with various AWT methods, such as drawString().

A JVM is required to view an applet. The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.

Advantage of Applet

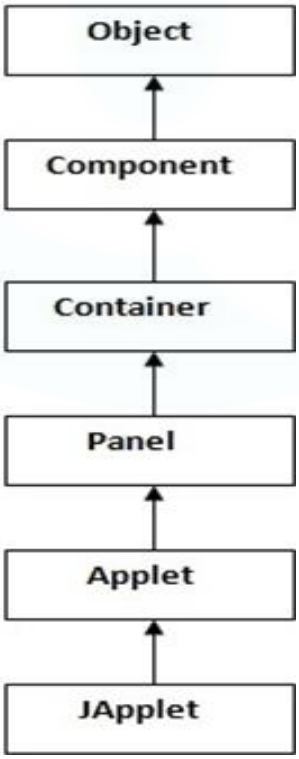
There are many advantages of applet. They are as follows:

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

Drawback of Applet

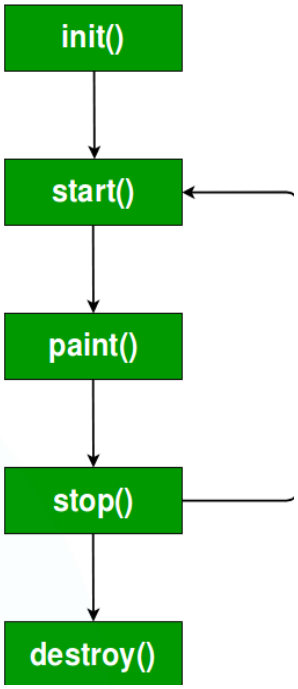
- Plugin is required at client browser to execute applet.

Hierarchy of Applet



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

Life cycle of an Applet



When an applet begins, the following methods are called, in this sequence:

1. init( )
2. start( )
3. paint( )

When an applet is terminated, the following sequence of method calls takes place:

1. stop( )
2. destroy( )

public void init(): is used to initialized the Applet.

The **init( )** method is the first method to be called. This is where you should initialize variables.

This method is called only once during the run time of your applet.

public void start(): is invoked after the init() method or browser is maximized. It is used to start the Applet.

The start( ) method is called after init( ).

It is also called to restart an applet after it has been stopped. Whereas init( ) is called once—the first time an applet is loaded—start( ) is called each time an applet's HTML document is displayed onscreen. So, if a user leaves a web page and comes back, the applet resumes execution at start( ).

public void paint(Graphics g): is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc. T

He paint( ) method is called each time your applet's output must be redrawn. This situation can occur for several reasons.

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.

paint( ) is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw its output, paint( ) is called.

The paint( ) method has one parameter of type Graphics. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required.

public void stop(): is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.

The stop( ) method is called when a web browser leaves the HTML document containing the applet—when it goes to another page, for example When stop( ) is called, the applet is probably running.

You should use stop( ) to suspend threads that don't need to run when the applet is not visible. You can restart them when start( ) is called if the user returns to the page.

public void destroy(): is used to destroy the Applet. It is invoked only once.

The destroy( ) method is called when the environment determines that your applet needs to be removed completely from memory. At this point, you should free up any resources the applet may be using.

The stop( ) method is always called before destroy( ).

Creating Applets

Creating Hello World applet:
// Save file as **HelloWorld.java**

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

public class HelloWorld extends Applet  // HelloWorld class extends Applet
{
   public void paint(Graphics g)    // Overriding paint() method @Override
   {
      g.drawString("Hello World", 20, 20);
   }
}
```

Explanation:

The above java program begins with two import statements. The first import statement imports the Applet class from applet package.

Every AWT-based(Abstract Window Toolkit) applet that you create must be a subclass (either directly or indirectly) of Applet class. The second statement imports the Graphics class from awt package.

The next line in the program declares the class HelloWorld. This class must be declared as public, because it will be accessed by code that is outside the program. Inside HelloWorld, **paint( )** is declared. This method is defined by the AWT and must be overridden by the applet.

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. In a Java window, the upper-left corner is location 0,0. The call to **drawString( )** in the applet causes the message "Hello World" 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( )**. In fact, most applets don't even have a **main( )** method. Instead, an applet begins execution when the name of its class is passed to an applet viewer or to a network browser.

Running the HelloWorld Applet:

After you enter the source code for HelloWorld.java, compile in the same way that you have been compiling java programs (using javac command). However running HelloWorld with the java command will generate an error because it is not an application.

There are two ways to run an applet

1. By html file (web browser).
2. By appletViewer tool (for testing purpose).

Using web browser:

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 click the html file.

// A Hello World Applet Using appletviewer
// HelloWorld.java

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

public class HelloWorld extends Applet  // HelloWorld class extends Applet
{
   public void paint(Graphics g)    // Overriding paint() method @Override
   {
      g.drawString("Hello World", 20, 20);
   }
}
```

Note: class must be public because its object is created by Java Plugin software that resides on the browser.

//myapplet.html

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

The width and height statements specify the dimensions of the display area used by the applet. After you create this html file, you can use it to execute the applet.

Using appletviewer :

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. Now Html file is not required.

// Save file as HelloWorld.java
// A Hello World Applet Using appletviewer

```
import java.applet.Applet;
import java.awt.Graphics;
/*
<applet code="HelloWorld" width=200 height=60></applet>
*/
 public class HelloWorld extends Applet // HelloWorld class extends Applet
{
   public void paint(Graphics g)
   {
      g.drawString("Hello World", 20, 20);
   }
}
```

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

```
c:\>javac HelloWorld.java
c:\>appletviewer HelloWorld.java
```

The HTML APPLET Tag

The APPLET tag is used to start an applet from both an HTML document and from an applet viewer. An applet viewer will execute each APPLET tag that it finds in a separate window, while web browsers like Netscape Navigator, Internet Explorer, and Hot Java will allow many applets on a single page.

The syntax for the standard APPLET tag is shown here. Bracketed items are optional.

< **APPLET**
[CODEBASE = codebaseURL]
CODE = appletFile
[ALT = alternateText]
[NAME= appletInstanceName]
WIDTH = pixels
HEIGHT = pixels
[ALIGN = alignment]
[VSPACE = pixels] [HSPACE = pixels]
>
[< PARAM NAME = Name1 VALUE =Value1>]
[< PARAM NAME = Name2 VALUE =Value2>]
…….
…….
</APPLET>

CODEBASE: CODEBASE is an optional attribute that specifies the base URL of the applet code, which is the directory that will be searched for the applet's executable class file (specified by the CODE tag).

CODE: CODE is a required attribute that gives the name of the file containing your applet's compiled .class file.

ALT: The ALT tag is an optional attribute used to specify a short text message that should be displayed if the browser understands the APPLET tag but can't currently run Java applets.

NAME: NAME is an optional attribute used to specify a name for the applet instance. Applets must be named in order for other applets on the same page to find them by name and communicate with them.

WIDTH AND HEIGHT: WIDTH and HEIGHT are required attributes that give the size (in pixels) of the applet display area.

ALIGN: ALIGN is an optional attribute that specifies the alignment of the applet.

This attribute is treated the same as the HTML IMG tag with these possible values: LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE, and ABSBOTTOM.

VSPACE AND HSPACE: These attributes are optional. VSPACE specifies the space, in pixels, above and below the applet.

HSPACE specifies the space, in pixels, on each side of the applet. They're treated the same as the IMG tag's VSPACE and HSPACE attributes.

PARAM NAME AND VALUE: The PARAM tag allows you to specify applet specific arguments in an HTML page. Applets access their attributes with the **getParameter( )** method.

Displaying Graphics in Applet

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

Commonly used methods of 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 of Graphics in applet:

```
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);
        }
```

```
}
```

//myapplet.html

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

Passing Parameters to Applets

The APPLET tag in HTML allows us to pass parameters to an applet. To retrieve a parameter, the **getParameter( )** method is used.

It returns the value of the specified parameter in the form of a **String** object. Thus, for numeric and **Boolean** values, we need to convert their string representations into their internal formats.

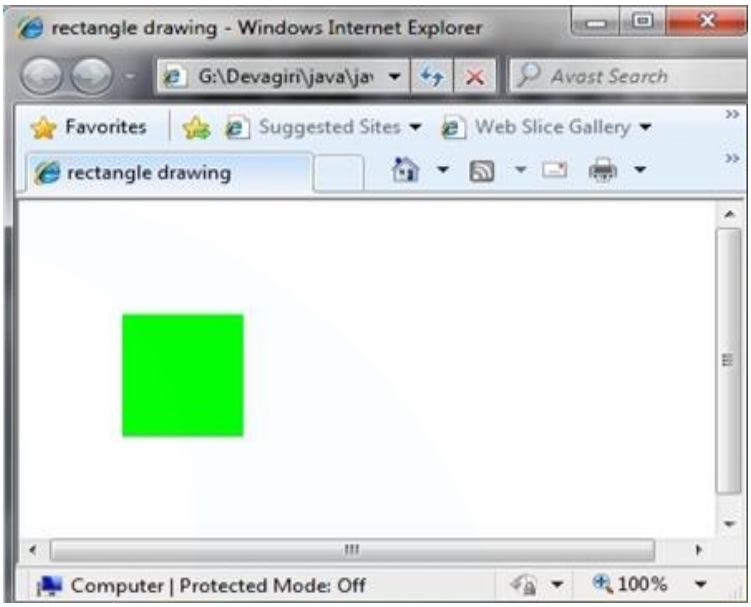
Here is an example that demonstrates passing parameters:

```
import java.awt.*;
import java.applet.*;
public class pgm17 extends Applet
{
        String str;
        int x1,x2,x3,x4;
        public void paint(Graphics g)
        {
                int x1=Integer.parseInt(getParameter("x1"));
                int x2=Integer.parseInt(getParameter("x2"));
                int x3=Integer.parseInt(getParameter("x3"));
                int x4=Integer.parseInt(getParameter("x4"));
                String str=getParameter("color");
                if(str.equals("blue"))
                        g.setColor(Color.blue);
                else if(str.equals("green"))
                        g.setColor(Color.green);
                else if(str.equals("red"))
                        g.setColor(Color.red);
                else if(str.equals("yellow"))
                        g.setColor(Color.yellow);
                g.fillRect(x1,x2,x3,x4);
        }
}
```

// pgm17.html

```
<html>
<head>
        <title> rectangle drawing</title>
</head>
<body>
<applet code="pgm17.class" width=400 height=200>
        <param name=x1 value=50>
        <param name=x2 value=60>
        <param name=x3 value=70>
        <param name=x4 value=80>
        <param name=color value="green">
</applet>
</body>
</html>
```

Output:



Event Handling

Events

Change in the state of an object is known as event i.e. event describes the change in state of source.

When user interacts with a GUI component, the interaction is known as an event

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.

Events are supported by the java.awt.event package

The java.awt.event package provides many event classes and Listener interfaces for event handling.

Types of Event

The events can be broadly classified into two categories:

- **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.

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

This mechanism has the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.

The delegation Event model

The event delegation model is based on the Event Source and Event Listeners.

The Delegation Event Model has the following key participants namely:

- **The Event Source** is any object which creates the message/event.

  The source is an object on which event occurs. Source is responsible for providing information of the occurred event to its handler. Java provide as with classes for source object.

- **Event Listener** is an object that receives the messages / events.

  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 processes the event and then returns.

  - In order to design a listener class we 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 the any if the predefined interfaces then your class can not act as a listener class for a source object.

- **Event handlers** are method that performs a task in response to an event is called an event handler

  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.

  In the delegation event model a source generates an event and sends it to one or more listeners. In this scheme, the listener simply waits until it receives an event. Once received, the listener processes the event and then returns.

  In event handling model, delegates act as intermediaries between objects that generate events and methods that handle those events.

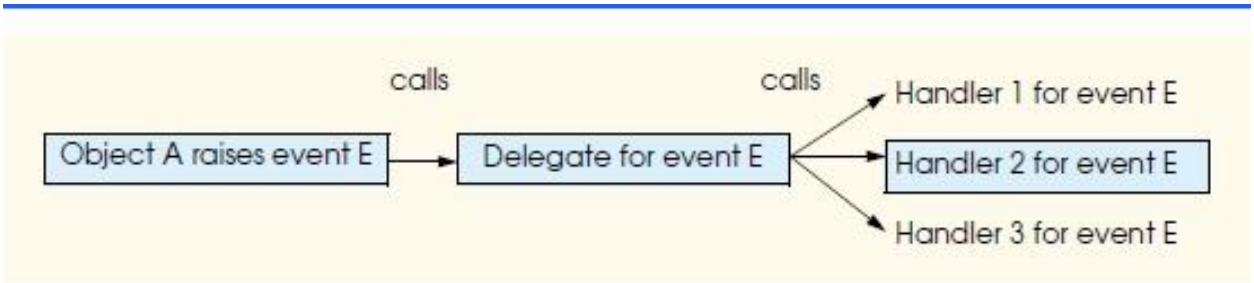


Fig. 12.5 Event-handling model using delegates.

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 listeners that want to receive them.

How Events are handled:

A source generates an Event and sends 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**.

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.

- 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. First, it must have been registered with one or more sources to receive notifications about specific types of events.
- The listener 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**.

Eg. 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.

Some methods related to Events are:

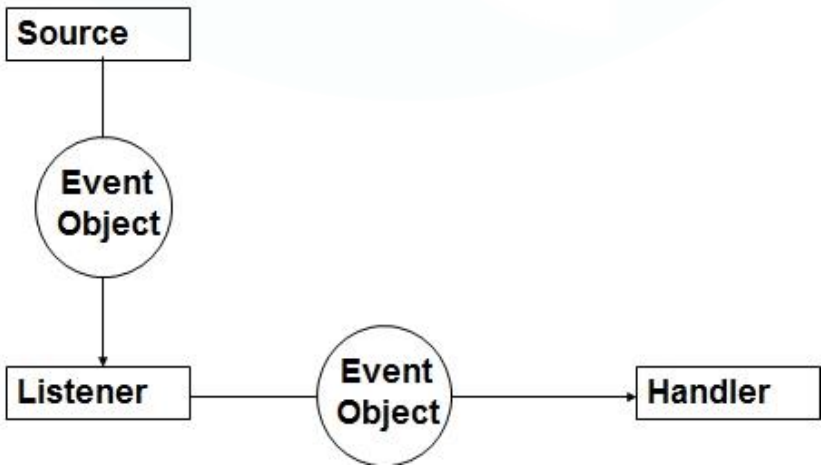
Object getSource()

This method is defined in the Event Object class. It returns the object that originated the event. So if the user has clicked a Push button, the method getSource returns the object corresponding to Push button is generated.

int getID()

This method returns the integer value corresponding to the type of event.

For example if the user has clicked on the mouse, the method getID returns the integer defined as MouseEvent.MOUSE_CLICKED.



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.

Event classes and Listener interfaces

Event Type	Event Source	Event Listener interface
ActionEvent	Button, List, MenuItem, TextField	ActionListener
AdjustmentEvent	Scrollbar	AdjustmentListener
ItemEvent	Choice, Checkbox, CheckboxMenuItem, List	ItemListener
TextEvent	TextArea, TextField	TextListener
ComponentEvent	Component	ComponentListener
ContainerEvent	Container	ContainerListener
FocusEvent	Component	FocusListener
KeyEvent	Component	KeyListener
MouseEvent	Component	MouseListener, MouseMotionListener
WindowEvent	Window	WindowListener

Event Listener Interfaces and corresponding methods which it defines	
Event Listener interface	**Event Listener Methods**
ActionListener	actionPerformed(ActionEvent evt)
AdjustmentListener	adjustmentValueChanged(AjustmentEvent evt)
ItemListener	itemStateChanged(ItemEvent evt)
TextListener	textValueChanged(TextEvent evt)
ComponentListener	componentHidden(ComponentEvent evt), componentMoved(ComponentEvent evt), componentResized(ComponentEvent evt), componentShown(ComponentEvent evt)
ContainerListener	componentAdded(ContainerEvent evt), componentRemoved(ContainerEvent evt)
FocusListener	focusGained(FocusEvent evt), focusLost(FocusEvent evt)
KeyListener	keyPressed(KeyEvent evt), keyReleased(KeyEvent evt), keyTyped(KeyEvent evt)
MouseListener	mouseClicked(MouseEvent evt), mouseEntered(MouseEvent evt), mouseExited(MouseEvent evt), mousePressed(MouseEvent evt), mouseReleased(MouseEvent evt)
MouseMotionListener	mouseDragged(MouseEvent evt), mouseMoved(MouseEvent evt)
WindowListener	windowActivated(WindowEvent evt), windowClosed(WindowEvent evt), windowClosing(WindowEvent evt), windowDeactivated(WindowEvent evt), windowDeiconified(WindowEvent evt), windowIconified(WindowEvent evt), windowOpened(WindowEvent evt)