UNIT IV AWT AND THREADS **9**

Applets: Basics of applets - Applet Architecture - Life cycle of an Applet – AWT : Event Handling-Delegation event Model.

Threads: Thread priority - Thread operations - Thread states – Thread Synchronization - Basics of Java Networking.

4.1 Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

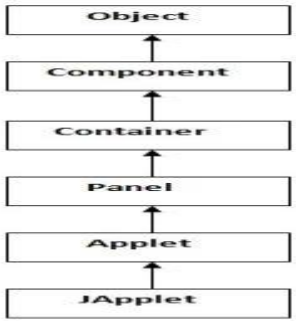
There are many advantages of applet. They are as follows:

- 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

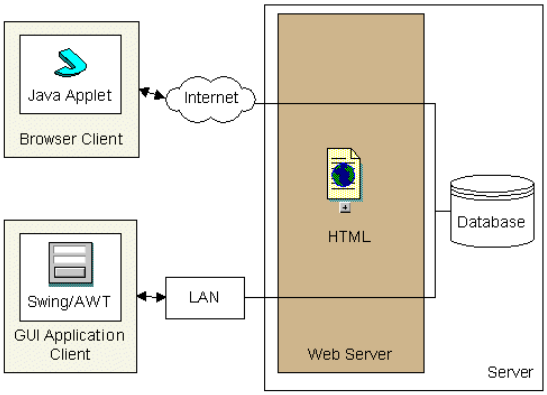
- Plugin is required at client browser to execute applet.

Hierarchy of Applet



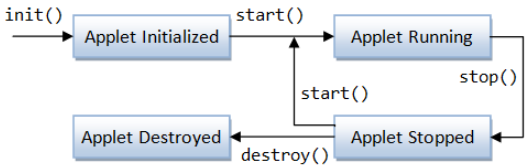
As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

4.2 Architecture of an Applet



4.3 Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.



Lifecycle methods for Applet:

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. **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.

Who is responsible to manage the life cycle of an applet?

Java Plug-in software.

How to run an Applet?

There are two ways to run an applet

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

Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

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

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

Simple example of Applet by appletviewer tool:

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

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

```
public class First extends Applet{
  public void paint(Graphics g){
g.drawString("welcome to applet",150,150);
}
  }
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

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

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

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>
```

Displaying Image in Applet

Applet is mostly used in games and animation. For this purpose image is required to be displayed. The java.awt.Graphics class provide a method drawImage() to display the image.

Syntax of drawImage() method:

1. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.

How to get the object of Image:

The java.applet.Applet class provides getImage() method that returns the object of Image. Syntax:

1. **public** Image getImage(URL u, String image){}

Other required methods of Applet class to display image:

1. **public URL getDocumentBase():** is used to return the URL of the document in which applet is embedded.
2. **public URL getCodeBase():** is used to return the base URL.

Example of displaying image in applet:

```
import java.awt.*;
import java.applet.*;
 public class DisplayImage extends Applet {
   Image picture;
   public void init() {
   picture = getImage(getDocumentBase(),"sonoo.jpg");
 }
    public void paint(Graphics g) {
   g.drawImage(picture, 30,30, this);
 }
     }
```

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

myapplet.html

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

Animation in Applet

Applet is mostly used in games and animation. For this purpose image is required to be moved.

Example of animation in applet:

```
import java.awt.*;
import java.applet.*;
public class AnimationExample extends Applet {

 Image picture;

 public void init() {
```

```
   picture =getImage(getDocumentBase(),"bike_1.gif");
  }

  public void paint(Graphics g) {
   for(int i=0;i<500;i++){
    g.drawImage(picture, i,30, this);

    try{Thread.sleep(100);}catch(Exception e){}
   }
  }
}
```

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

myapplet.html

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

EventHandling in Applet

As we perform event handling in AWT or Swing, we can perform it in applet also. Let's see the simple example of event handling in applet that prints a message by click on the button.

Example of EventHandling in applet:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class EventApplet extends Applet implements ActionListener{
Button b;
TextField tf;
  public void init(){
tf=new TextField();
tf.setBounds(30,40,150,20);
  b=new Button("Click");
b.setBounds(80,150,60,50);
  add(b);add(tf);
b.addActionListener(this);
  setLayout(null);
}
   public void actionPerformed(ActionEvent e){
  tf.setText("Welcome");
 }
}
```

In the above example, we have created all the controls in init() method because it is invoked only once.

myapplet.html

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

EventHandling in Applet

As we perform event handling in AWT or Swing, we can perform it in applet also. Let's see the simple example of event handling in applet that prints a message by click on the button.

Example of EventHandling in applet:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class EventApplet extends Applet implements ActionListener{
Button b;
TextField tf;
  public void init(){
tf=new TextField();
tf.setBounds(30,40,150,20);
  b=new Button("Click");
b.setBounds(80,150,60,50);
 add(b);add(tf);
b.addActionListener(this);
setLayout(null);
}
   public void actionPerformed(ActionEvent e){
  tf.setText("Welcome");
 }
}
```

In the above example, we have created all the controls in init() method because it is invoked only once.

myapplet.html

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

Digital clock in Applet

Digital clock can be created by using the Calendar and SimpleDateFormat class. Let's see the simple example:

Example of Digital clock in Applet:

```
import java.applet.*;
import java.awt.*;
import java.util.*;
import java.text.*;
  public class DigitalClock extends Applet implements Runnable {
     Thread t = null;
   int hours=0, minutes=0, seconds=0;
   String timeString = "";
     public void init() {
      setBackground( Color.green);
   }
     public void start() {
        t = new Thread( this );
        t.start();
   }


   public void run() {
     try {
```

```
        while (true) {

          Calendar cal = Calendar.getInstance();
          hours = cal.get( Calendar.HOUR_OF_DAY );
          if ( hours > 12 ) hours -= 12;
          minutes = cal.get( Calendar.MINUTE );
          seconds = cal.get( Calendar.SECOND );

          SimpleDateFormat formatter = new SimpleDateFormat("hh:mm:ss");
          Date date = cal.getTime();
          timeString = formatter.format( date );

          repaint();
          t.sleep( 1000 );  // interval given in milliseconds
        }
      }
      catch (Exception e) { }
    }


   public void paint( Graphics g ) {
      g.setColor( Color.blue );
      g.drawString( timeString, 50, 50 );
    }
}
```

In the above example, getX() and getY() method of MouseEvent is used to get the current x-axis and y-axis. The getGraphics() method of Component class returns the object of Graphics.

myapplet.html

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

Parameter in Applet

We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named getParameter(). Syntax:

```
public String getParameter(String parameterName)
```

Example of using parameter in Applet:

```
import java.applet.Applet;
import java.awt.Graphics;
  public class UseParam extends Applet{
  public void paint(Graphics g){
String str=getParameter("msg");
g.drawString(str,50, 50);
}
  }
```

myapplet.html

```
<html>
<body>
<applet code="UseParam.class" width="300" height="300">
<param name="msg" value="Welcome to applet">
</applet>
</body>
</html>
```

Applet Communication

java.applet.AppletContext class provides the facility of communication between applets. We provide the name of applet through the HTML file. It provides getApplet() method that returns the object of Applet. Syntax:

```
public Applet getApplet(String name){}
```

Example of Applet Communication

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class ContextApplet extends Applet implements ActionListener{
Button b;
  public void init(){
b=new Button("Click");
b.setBounds(50,50,60,50);
 add(b);
b.addActionListener(this);
}
  public void actionPerformed(ActionEvent e){
  AppletContext ctx=getAppletContext();
Applet a=ctx.getApplet("app2");
a.setBackground(Color.yellow);
}               }
```

myapplet.html

```
<html>
<body>
<applet code="ContextApplet.class" width="150" height="150" name="app1">
</applet>
  <applet code="First.class" width="150" height="150" name="app2">
</applet>
</body>
</html>
```

4.4 Java AWT Tutorial

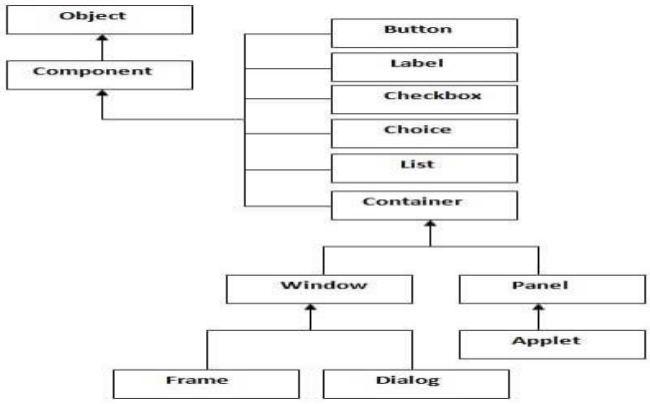
Java AWT (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



Container

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

Window

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

Panel

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

Frame

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

Useful Methods of Component class

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

Java AWT Example

To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

- By extending Frame class (inheritance)
- By creating the object of Frame class (association)

AWT Example by Inheritance

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

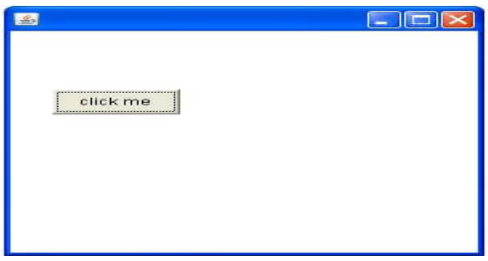
```
import java.awt.*;
class First extends Frame{
First(){
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position
add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
setLayout(null);//no layout manager
setVisible(true);//now frame will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

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

AWT Example by Association

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

```
import java.awt.*;
class First2{
First2(){
Frame f=new Frame();
Button b=new Button("click me");
b.setBounds(30,50,80,30);
f.add(b);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[]){
First2 f=new First2();  }}
```



Event and Listener (Java Event Handling)

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

Java Event classes and Listener interfaces

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

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
  - public void addActionListener(ActionListener a){}
- **MenuItem**
  - public void addActionListener(ActionListener a){}
- **TextField**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- **TextArea**
  - public void addTextListener(TextListener a){}
- **Checkbox**
  - public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

Java Event Handling Code

We can put the event handling code into one of the following places:

1. Within class
2. Other class
3. Anonymous class

Java event handling by implementing ActionListener

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener{
TextField tf;
AEvent(){
  //create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
  //register listener
b.addActionListener(this);//passing current instance
  //add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
public static void main(String args[]){
new AEvent();
}
}
```

public void setBounds(int xaxis, int yaxis, int width, int height); have been used in the above example that sets the position of the component it may be button, textfield etc.

2) Java event handling by outer class

```
import java.awt.*;
import java.awt.event.*;
class AEvent2 extends Frame{
TextField tf;
AEvent2(){
//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
//register listener
Outer o=new Outer(this);
b.addActionListener(o);//passing outer class instance
//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[]){
new AEvent2();
}
}
import java.awt.event.*;
class Outer implements ActionListener{
AEvent2 obj;
Outer(AEvent2 obj){
this.obj=obj;
}
public void actionPerformed(ActionEvent e){
obj.tf.setText("welcome");
}
}
```

3) Java event handling by anonymous class

```
import java.awt.*;
import java.awt.event.*;
class AEvent3 extends Frame{
TextField tf;
AEvent3(){
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(50,120,80,30);

b.addActionListener(new ActionListener(){
public void actionPerformed(){
tf.setText("hello");
}
});
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[]){
new AEvent3();
}
}
```

Java AWT Button

The button class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed.

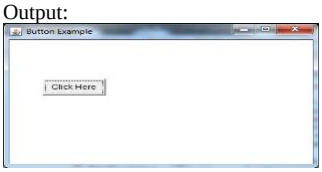
AWT Button Class declaration

```
public class Button extends Component implements Accessible
```

Java AWT Button Example

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

Output:

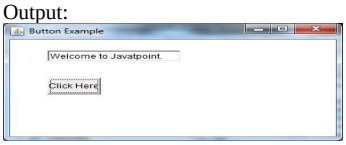


Java AWT Button Example with ActionListener

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

tf.setText("Welcome to Javatpoint.");
} });
f.add(b);f.add(tf);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
} }
```

Output:



Java AWT TextField

The object of a TextField class is a text component that allows the editing of a single line text. It inherits TextComponent class.

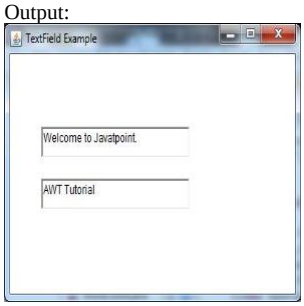
AWT TextField Class Declaration

```
public class TextField extends TextComponent
```

Java AWT TextField Example

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

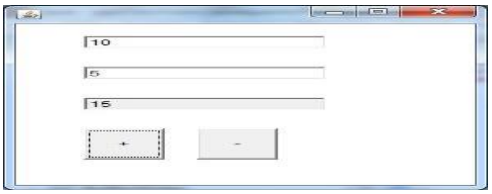
Output:



Java AWT TextField Example with ActionListener

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

Output:



Java AWT Checkbox

The Checkbox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a Checkbox changes its state from "on" to "off" or from "off" to "on".

AWT Checkbox Class Declaration

```
public class Checkbox extends Component implements ItemSelectable, Accessible
```

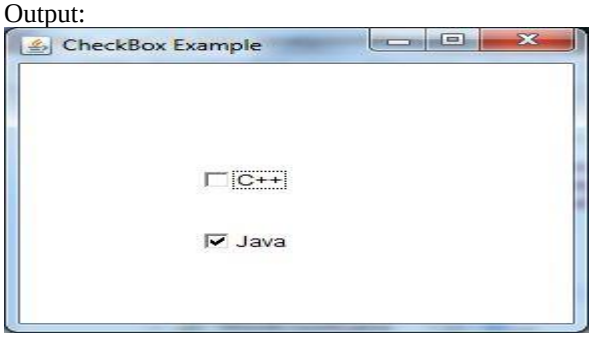
Java AWT Choice

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits Component class.

Java AWT Checkbox Example

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

Output:



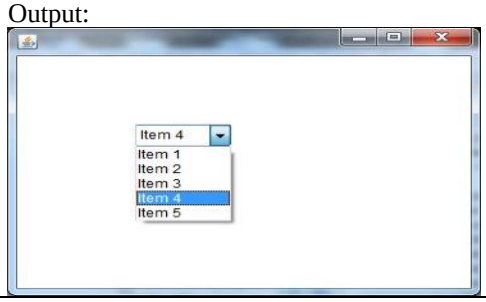
AWT Choice Class Declaration

```
public class Choice extends Component implements ItemSelectable, Accessible
```

Java AWT Choice Example

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

Output:



Java AWT List

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

AWT List class Declaration

1. `public class List extends Component implements ItemSelectable, Accessible`

Java AWT Canvas

The Canvas control represents a blank rectangular area where the application can draw or trap input events from the user. It inherits the Component class.

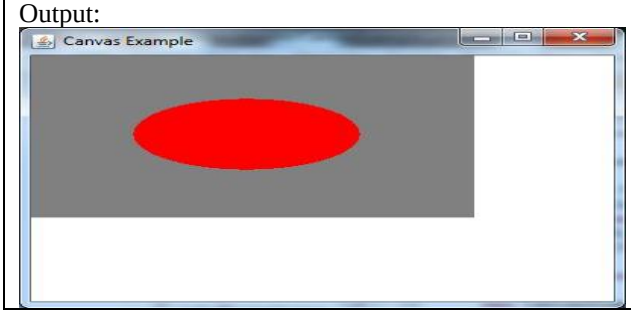
AWT Canvas class Declaration

1. `public class Canvas extends Component implements Accessible`

Java AWT Canvas Example

```
    import java.awt.*;
    public class CanvasExample
{
     public CanvasExample()
 {
   Frame f= new Frame("Canvas Example");
   f.add(new MyCanvas());
   f.setLayout(null);
   f.setSize(400, 400);
   f.setVisible(true);
 }
 public static void main(String args[])
 {
   new CanvasExample();
 }
}
class MyCanvas extends Canvas
{
      public MyCanvas() {
       setBackground (Color.GRAY);
       setSize(300, 200);
    }
 public void paint(Graphics g)
 {
   g.setColor(Color.red);
   g.fillOval(75, 75, 150, 75);
 }
}
```

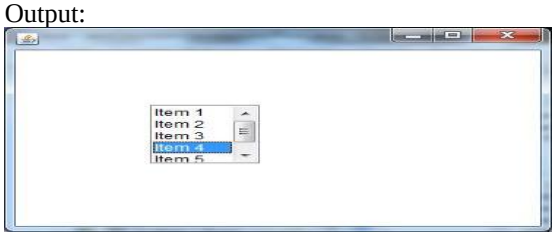
Output:



Java AWT List Example

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

Output:



Java AWT MenuItem and Menu

The object of MenuItem class adds a simple labeled menu item on menu. The items used in a menu must belong to the MenuItem or any of its subclass.

The object of Menu class is a pull down menu component which is displayed on the menu bar. It inherits the MenuItem class.

AWT MenuItem class declaration

1. **public class** MenuItem **extends** MenuComponent **implements** Accessible

AWT Menu class declaration

1. **public class** Menu **extends** MenuItem **implements** MenuContainer, Accessible

Java AWT Panel

The Panel is a simplest container class. It provides space in which an application can attach any other component. It inherits the Container class.

It doesn't have title bar.

AWT Panel class declaration

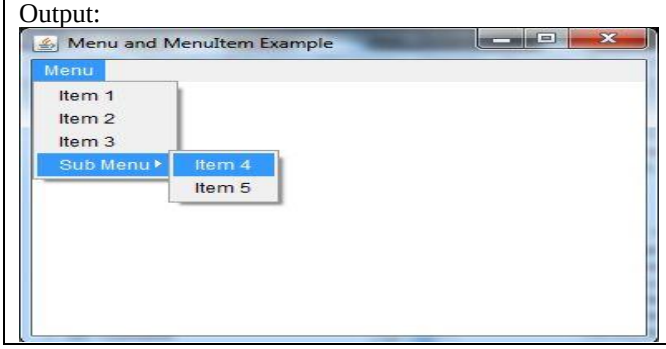
public class Panel **extends** Container **implements** Accessible

Java AWT MenuItem and Menu Example

```
import java.awt.*;
class MenuExample
{
    MenuExample(){
        Frame f= new Frame("Menu and MenuItem Example");

        MenuBar mb=new MenuBar();
        Menu menu=new Menu("Menu");
        Menu submenu=new Menu("Sub Menu");
        MenuItem i1=new MenuItem("Item 1");
        MenuItem i2=new MenuItem("Item 2");
        MenuItem i3=new MenuItem("Item 3");
        MenuItem i4=new MenuItem("Item 4");
        MenuItem i5=new MenuItem("Item 5");
        menu.add(i1);
        menu.add(i2);
        menu.add(i3);
        submenu.add(i4);
        submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
}
    public static void main(String args[])
{
new MenuExample();  }
```

Output:

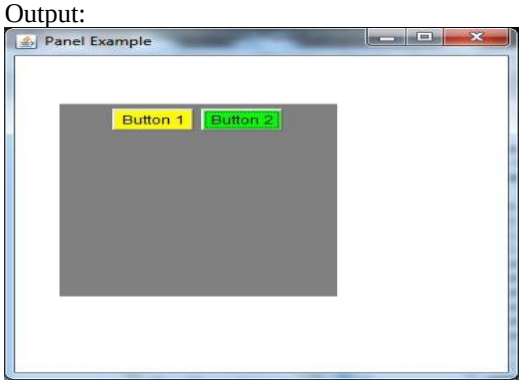


Java AWT Panel Example

```
import java.awt.*;
public class PanelExample {
    PanelExample()
        {
        Frame f= new Frame("Panel Example");

        Panel panel=new Panel();
        panel.setBounds(40,80,200,200);
        panel.setBackground(Color.gray);
        Button b1=new Button("Button 1");
        b1.setBounds(50,100,80,30);
        b1.setBackground(Color.yellow);
        Button b2=new Button("Button 2");
        b2.setBounds(100,100,80,30);
        b2.setBackground(Color.green);
        panel.add(b1); panel.add(b2);
        f.add(panel);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
        }
        public static void main(String args[])
        {
        new PanelExample();
        }
    }
```

Output:



Java AWT Toolkit

Toolkit class is the abstract superclass of every implementation in the Abstract Window Toolkit. Subclasses of Toolkit are used to bind various components. It inherits Object class.

AWT Toolkit class declaration

```
public abstract class Toolkit extends Object
```

Java AWT Toolkit Example

```
import java.awt.*;
public class ToolkitExample {
  public static void main(String[] args) {
     Toolkit t = Toolkit.getDefaultToolkit();
     System.out.println("Screen resolution = " + t.getScreenResolution());
```

```
        Dimension d = t.getScreenSize();
        System.out.println("Screen width = " + d.width);
        System.out.println("Screen height = " + d.height);
    }
}
```

Output:

```
Screen resolution = 96
Screen width = 1366
Screen height = 768
```

Java ActionListener Interface

The Java ActionListener is notified whenever you click on the button or menu item. It is notified against ActionEvent. The ActionListener interface is found in java.awt.event package. It has only one method: actionPerformed().

actionPerformed() method

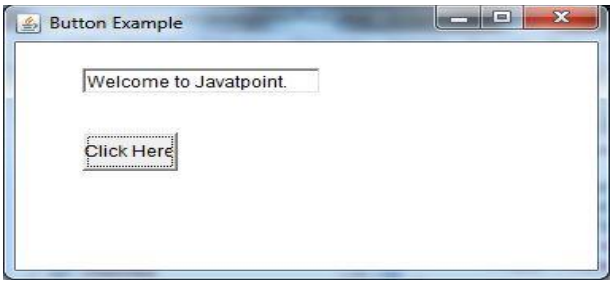
The actionPerformed() method is invoked automatically whenever you click on the registered component.

```
public abstract void actionPerformed(ActionEvent e);
```

Java ActionListener Example: On Button click

```
import java.awt.*;
import java.awt.event.*;
public class ActionListenerExample {
public static void main(String[] args) {
    Frame f=new Frame("ActionListener Example");
    final TextField tf=new TextField();
    tf.setBounds(50,50, 150,20);
    Button b=new Button("Click Here");
    b.setBounds(50,100,60,30);
     b.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
          tf.setText("Welcome to Javatpoint.");
    }
    });
    f.add(b);f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Output:



Java MouseListener Interface

The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.

Methods of MouseListener interface

The signature of 5 methods found in MouseListener interface are given below:

```
public abstract void mouseClicked(MouseEvent e);
public abstract void mouseEntered(MouseEvent e);
```

```
public abstract void mouseExited(MouseEvent e);
public abstract void mousePressed(MouseEvent e);
public abstract void mouseReleased(MouseEvent e);
```

Java MouseListener Example

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

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

Output:

Java ItemListener Interface

The Java ItemListener is notified whenever you click on the checkbox. It is notified against ItemEvent. The ItemListener interface is found in java.awt.event package. It has only one method: itemStateChanged().

itemStateChanged() method

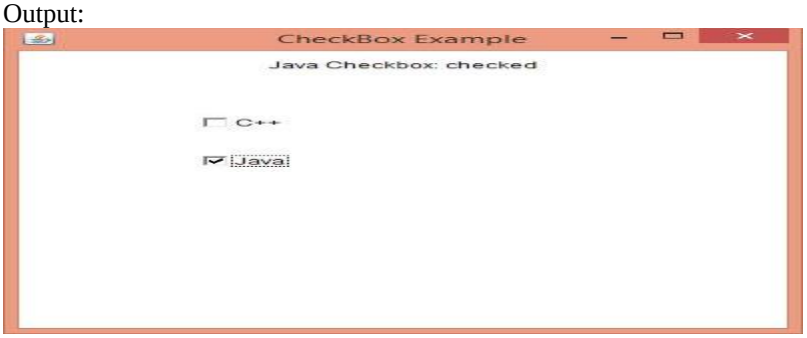
The itemStateChanged() method is invoked automatically whenever you click or unclick on the registered checkbox component.

```
public abstract void itemStateChanged(ItemEvent e);
```

Java ItemListener Example

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

Output:



Java WindowListener Interface

The Java WindowListener is notified whenever you change the state of window. It is notified against WindowEvent. The WindowListener interface is found in java.awt.event package. It has three methods.

Methods of WindowListener interface

The signature of 7 methods found in WindowListener interface are given below:

1. **public abstract void** windowActivated(WindowEvent e);
2. **public abstract void** windowClosed(WindowEvent e);
3. **public abstract void** windowClosing(WindowEvent e);
4. **public abstract void** windowDeactivated(WindowEvent e);
5. **public abstract void** windowDeiconified(WindowEvent e);

6. **public abstract void** windowIconified(WindowEvent e);
7. **public abstract void** windowOpened(WindowEvent e);

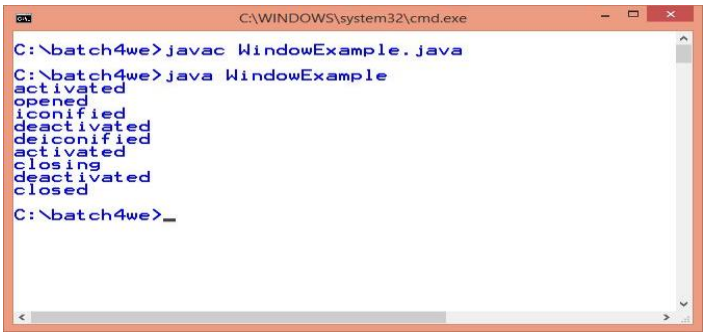
Java WindowListener Example

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

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

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

Output:



Java Adapter Classes

Java adapter classes *provide the default implementation of listener interfaces*. 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 **java.awt.event**, **java.awt.dnd** and **javax.swing.event** packages. The Adapter classes with their corresponding listener interfaces are given below.

java.awt.event Adapter classes

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

java.awt.dnd Adapter classes

Adapter class	Listener interface
DragSourceAdapter	DragSourceListener
DragTargetAdapter	DragTargetListener

javax.swing.event Adapter classes

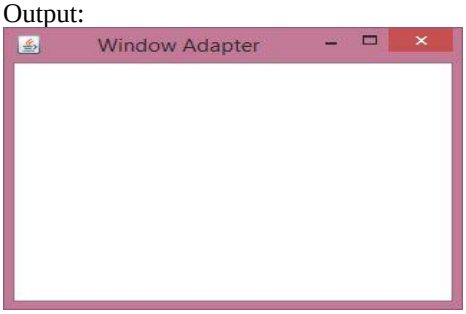
Adapter class	Listener interface
MouseInputAdapter	MouseInputListener
InternalFrameAdapter	InternalFrameListener

Java WindowAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class AdapterExample{
   Frame f;
   AdapterExample(){
      f=new Frame("Window Adapter");
      f.addWindowListener(new WindowAdapter(){
         public void windowClosing(WindowEvent e) {
            f.dispose();
         }
      });
            f.setSize(400,400);
      f.setLayout(null);
```

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

Output:



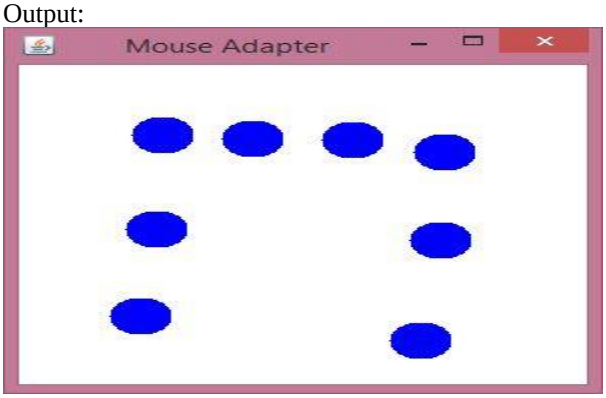
Java MouseAdapter Example

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

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

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

Output:



How to close AWT Window in Java

We can close the AWT Window or Frame by calling *dispose()* or *System.exit()* inside windowClosing() method. The windowClosing() method is found in **WindowListener** interface and **WindowAdapter** class.

The WindowAdapter class implements WindowListener interfaces. It provides the default implementation of all the 7 methods of WindowListener interface. To override the windowClosing() method, you can either use WindowAdapter class or WindowListener interface.

If you implement the WindowListener interface, you will be forced to override all the 7 methods of WindowListener interface. So it is better to use WindowAdapter class.

Different ways to override windowClosing() method

There are many ways to override windowClosing() method:

- By anonymous class
- By inheriting WindowAdapter class
- By implementing WindowListener interface

Close AWT Window Example 1: Anonymous class

```
import java.awt.*;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
public class WindowExample extends Frame{
   WindowExample(){
      addWindowListener(new WindowAdapter(){
         public void windowClosing(WindowEvent e) {
            dispose();
         }
      });
      setSize(400,400);
      setLayout(null);
      setVisible(true);
   }
public static void main(String[] args) {
   new WindowExample();
}
```

Output:

EVENTS	SOURCE	LISTENERS
Action Event	Button, List, Menultem, Text field	ActionListener
Component Event	Component	Component Listener
Focus Event	Component	FocusListener
Item Event	Checkbox, CheckboxMenultem, Choice, List	ItemListener
Key Event	when input is received from keyboard	KeyListener
Text Event	Text Component	TextListener
Window Event	Window	WindowListener
Mouse Event	Mouse related event	MouseListener

Threads: Thread priority - Thread operations - Thread states – Thread Synchronization - Basics of Java Networking.

4.6 MultiThreading

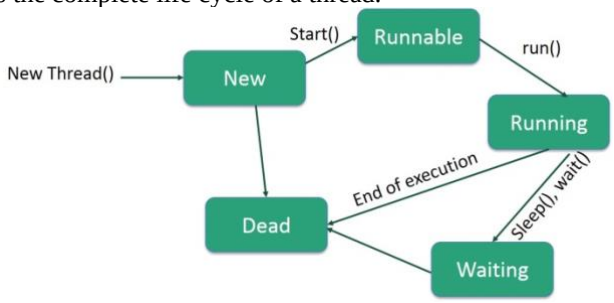
Java is a *multi-threaded programming language* which means we can develop multi-threaded program using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

By definition, multitasking is when multiple processes share common processing resources such as a CPU. Multi-threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

4. 7 Life Cycle of a Thread

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread.



Following are the stages of the life cycle −

- **New** − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a **born thread**.
- **Runnable** − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting** − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed Waiting** − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated (Dead)** − A runnable thread enters the terminated state when it completes its task or otherwise terminates.

4.8 Thread Priorities

Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled. Java thread priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5).

Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and are very much platform dependent.

Create a Thread by Implementing a Runnable Interface

If your class is intended to be executed as a thread then you can achieve this by implementing a **Runnable** interface. You will need to follow three basic steps −

Step 1

As a first step, you need to implement a run() method provided by a **Runnable** interface. This method provides an entry point for the thread and you will put your complete business logic inside this method. Following is a simple syntax of the run() method −

```
public void run( )
```

Step 2

As a second step, you will instantiate a **Thread** object using the following constructor −

```
Thread(Runnable threadObj, String threadName);
```

Where, *threadObj* is an instance of a class that implements the **Runnable**interface and **threadName** is the name given to the new thread.

Step 3

Once a Thread object is created, you can start it by calling **start()** method, which executes a call to run( ) method. Following is a simple syntax of start() method −

```
void start();
```

Example

Here is an example that creates a new thread and starts running it −

```
class RunnableDemo implements Runnable {
  private Thread t;
  private String threadName;

  RunnableDemo( String name) {
    threadName = name;
    System.out.println("Creating " +  threadName );
  }

  public void run() {
    System.out.println("Running " +  threadName );
    try {
      for(int i = 4; i > 0; i--) {
        System.out.println("Thread: " + threadName + ", " + i);
        // Let the thread sleep for a while.
        Thread.sleep(50);
      }
    } catch (InterruptedException e) {
      System.out.println("Thread " +  threadName + " interrupted.");
    }
    System.out.println("Thread " +  threadName + " exiting.");
  }

  public void start () {
    System.out.println("Starting " +  threadName );
    if (t == null) {
      t = new Thread (this, threadName);
      t.start ();
    }
  }
}

public class TestThread {

  public static void main(String args[]) {
    RunnableDemo R1 = new RunnableDemo( "Thread-1");
    R1.start();

    RunnableDemo R2 = new RunnableDemo( "Thread-2");
    R2.start();
  }
 }
```

This will produce the following result −

Output

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

Create a Thread by Extending a Thread Class

The second way to create a thread is to create a new class that extends **Thread** class using the following two simple steps. This approach provides more flexibility in handling multiple threads created using available methods in Thread class.

Step 1

You will need to override **run( )** method available in Thread class. This method provides an entry point for the thread and you will put your complete business logic inside this method. Following is a simple syntax of run() method −

```
public void run( )
```

Step 2

Once Thread object is created, you can start it by calling **start()** method, which executes a call to run( ) method. Following is a simple syntax of start() method −

```
void start( );
```

Example

Here is the preceding program rewritten to extend the Thread −

```
class ThreadDemo extends Thread {
  private Thread t;
  private String threadName;
    ThreadDemo( String name) {
    threadName = name;
    System.out.println("Creating " +  threadName );
  }

  public void run() {
    System.out.println("Running " +  threadName );
    try {
      for(int i = 4; i > 0; i--) {
        System.out.println("Thread: " + threadName + ", " + i);
        // Let the thread sleep for a while.
        Thread.sleep(50);
      }
    } catch (InterruptedException e) {
      System.out.println("Thread " +  threadName + " interrupted.");
    }
    System.out.println("Thread " +  threadName + " exiting.");  }
```

```
    public void start () {
    System.out.println("Starting " +  threadName );
    if (t == null) {
      t = new Thread (this, threadName);
      t.start ();
    }
  }
}

public class TestThread {

  public static void main(String args[]) {
    ThreadDemo T1 = new ThreadDemo( "Thread-1");
    T1.start();

    ThreadDemo T2 = new ThreadDemo( "Thread-2");
    T2.start();
  }
}This will produce the following result −
Output
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

Thread Methods

Following is the list of important methods available in the Thread class.

Sr.No.	Method & Description
1	**public void start()** Starts the thread in a separate path of execution, then invokes the run() method on this Thread object.
2	**public void run()** If this Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object.
3	**public final void setName(String name)** Changes the name of the Thread object. There is also a getName() method for retrieving the name.
4	**public final void setPriority(int priority)** Sets the priority of this Thread object. The possible values are between 1 and 10.

5	**public final void setDaemon(boolean on)** A parameter of true denotes this Thread as a daemon thread.
6	**public final void join(long millisec)** The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.
7	**public void interrupt()** Interrupts this thread, causing it to continue execution if it was blocked for any reason.
8	**public final boolean isAlive()** Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

The previous methods are invoked on a particular Thread object. The following methods in the Thread class are static. Invoking one of the static methods performs the operation on the currently running thread.

Sr.No.	Method & Description
1	**public static void yield()** Causes the currently running thread to yield to any other threads of the same priority that are waiting to be scheduled.
2	**public static void sleep(long millisec)** Causes the currently running thread to block for at least the specified number of milliseconds.
3	**public static boolean holdsLock(Object x)** Returns true if the current thread holds the lock on the given Object.
4	**public static Thread currentThread()** Returns a reference to the currently running thread, which is the thread that invokes this method.
5	**public static void dumpStack()** Prints the stack trace for the currently running thread, which is useful when debugging a multithreaded application.

Example

The following ThreadClassDemo program demonstrates some of these methods of the Thread class. Consider a class **DisplayMessage** which implements **Runnable** −

```
// File Name : DisplayMessage.java
// Create a thread to implement Runnable

public class DisplayMessage implements Runnable {
   private String message;

   public DisplayMessage(String message) {
      this.message = message;
   }

   public void run() {
      while(true) {
         System.out.println(message);
      }
   }
}
```

Following is another class which extends the Thread class −

```
// File Name : GuessANumber.java
// Create a thread to extentd Thread

public class GuessANumber extends Thread {
   private int number;
   public GuessANumber(int number) {
      this.number = number;
   }
      public void run() {
      int counter = 0;
      int guess = 0;
      do {
         guess = (int) (Math.random() * 100 + 1);
         System.out.println(this.getName() + " guesses " + guess);
         counter++;
      } while(guess != number);
      System.out.println("** Correct!" + this.getName() + "in" + counter + "guesses.**");
   }
}Following is the main program, which makes use of the above-defined classes –
// File Name : ThreadClassDemo.java
public class ThreadClassDemo {
   public static void main(String [] args) {
      Runnable hello = new DisplayMessage("Hello");
      Thread thread1 = new Thread(hello);
      thread1.setDaemon(true);
      thread1.setName("hello");
      System.out.println("Starting hello thread...");
      thread1.start();

      Runnable bye = new DisplayMessage("Goodbye");
      Thread thread2 = new Thread(bye);
      thread2.setPriority(Thread.MIN_PRIORITY);
      thread2.setDaemon(true);
      System.out.println("Starting goodbye thread...");
      thread2.start();

      System.out.println("Starting thread3...");
      Thread thread3 = new GuessANumber(27);
      thread3.start();
      try {
         thread3.join();
      } catch (InterruptedException e) {
         System.out.println("Thread interrupted.");
      }
      System.out.println("Starting thread4...");
      Thread thread4 = new GuessANumber(75);

      thread4.start();
      System.out.println("main() is ending...");
   }
}
```

This will produce the following result. You can try this example again and again and you will get a different result every time.

Output

```
Starting hello thread...
Starting goodbye thread...
```

```
Hello
Hello
Hello
Hello
Hello
Hello
Goodbye
Goodbye
Goodbye
Goodbye
Goodbye
```

What is thread synchronization?

When we start two or more threads within a program, there may be a situation when multiple threads try to access the same resource and finally they can produce unforeseen result due to concurrency issues. For example, if multiple threads try to write within a same file then they may corrupt the data because one of the threads can override data or while one thread is opening the same file at the same time another thread might be closing the same file.

So there is a need to synchronize the action of multiple threads and make sure that only one thread can access the resource at a given point in time. This is implemented using a concept called **monitors**. Each object in Java is associated with a monitor, which a thread can lock or unlock. Only one thread at a time may hold a lock on a monitor.

Java programming language provides a very handy way of creating threads and synchronizing their task by using **synchronized** blocks. You keep shared resources within this block. Following is the general form of the synchronized statement −

Syntax

```
synchronized(objectidentifier) {
   // Access shared variables and other shared resources
}
```

Here, the **objectidentifier** is a reference to an object whose lock associates with the monitor that the synchronized statement represents. Now we are going to see two examples, where we will print a counter using two different threads. When threads are not synchronized, they print counter value which is not in sequence, but when we print counter by putting inside synchronized() block, then it prints counter very much in sequence for both the threads.

Multithreading Example without Synchronization

Here is a simple example which may or may not print counter value in sequence and every time we run it, it produces a different result based on CPU availability to a thread.

Example

```
class PrintDemo {
  public void printCount() {
    try {
      for(int i = 5; i > 0; i--) {
        System.out.println("Counter   ---   "  + i );
      }
    } catch (Exception e) {
      System.out.println("Thread  interrupted.");
    }
  }
}
class ThreadDemo extends Thread {
  private Thread t;
  private String threadName;
  PrintDemo  PD;

  ThreadDemo( String name,  PrintDemo pd) {
    threadName = name;
    PD = pd;
```

```
  }
    public void run() {
    PD.printCount();
    System.out.println("Thread " +  threadName + " exiting.");
  }
  public void start () {
    System.out.println("Starting " +  threadName );
    if (t == null) {
      t = new Thread (this, threadName);
      t.start ();
    }
  }
}
public class TestThread {
  public static void main(String args[]) {
    PrintDemo PD = new PrintDemo();
    ThreadDemo T1 = new ThreadDemo( "Thread - 1 ", PD );
    ThreadDemo T2 = new ThreadDemo( "Thread - 2 ", PD );
    T1.start();
    T2.start();
    // wait for threads to end
      try {
      T1.join();
      T2.join();
    } catch ( Exception e) {
      System.out.println("Interrupted");
    }
  }
}
```

This produces a different result every time you run this program −

Output

```
Starting Thread - 1
Starting Thread - 2
Counter   ---   5
Counter   ---   4
Counter   ---   3
Counter   ---   5
Counter   ---   2
Counter   ---   1
Counter   ---   4
Thread Thread - 1  exiting.
Counter   ---   3
Counter   ---   2
Counter   ---   1
Thread Thread - 2  exiting.
```

Multithreading Example with Synchronization

Here is the same example which prints counter value in sequence and every time we run it, it produces the same result.

Example

```
class PrintDemo {
  public void printCount() {
    try {
      for(int i = 5; i > 0; i--) {
        System.out.println("Counter   ---   "  + i );
      }
    } catch (Exception e) {
```

```
        System.out.println("Thread  interrupted.");
      }
    }
}

class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;
   PrintDemo  PD;

   ThreadDemo( String name,  PrintDemo pd) {
      threadName = name;
      PD = pd;
   }

   public void run() {
      synchronized(PD) {
         PD.printCount();
      }
      System.out.println("Thread " +  threadName + " exiting.");
   }

   public void start () {
      System.out.println("Starting " +  threadName );
      if (t == null) {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}

public class TestThread {

   public static void main(String args[]) {
      PrintDemo PD = new PrintDemo();

      ThreadDemo T1 = new ThreadDemo( "Thread - 1 ", PD );
      ThreadDemo T2 = new ThreadDemo( "Thread - 2 ", PD );

      T1.start();
      T2.start();

      // wait for threads to end
      try {
         T1.join();
         T2.join();
      } catch ( Exception e) {
         System.out.println("Interrupted");
      }
   }
}
```

This produces the same result every time you run this program −

Output

```
Starting Thread - 1
Starting Thread - 2
Counter   ---   5
```

```
Counter   ---   4
Counter   ---   3
Counter   ---   2
Counter   ---   1
Thread Thread - 1  exiting.
Counter   ---   5
Counter   ---   4
Counter   ---   3
Counter   ---   2
Counter   ---   1
Thread Thread - 2  exiting.
```

Core Java provides complete control over multithreaded program. You can develop a multithreaded program which can be suspended, resumed, or stopped completely based on your requirements. There are various static methods which you can use on thread objects to control their behavior. Following table lists down those methods −

Sr.No.	Method & Description
1	**public void suspend()** This method puts a thread in the suspended state and can be resumed using resume() method.
2	**public void stop()** This method stops a thread completely.
3	**public void resume()** This method resumes a thread, which was suspended using suspend() method.
4	**public void wait()** Causes the current thread to wait until another thread invokes the notify().
5	**public void notify()** Wakes up a single thread that is waiting on this object's monitor.

Be aware that the latest versions of Java has deprecated the usage of suspend( ), resume( ), and stop( ) methods and so you need to use available alternatives.

Example

```
class RunnableDemo implements Runnable {
  public Thread t;
  private String threadName;
  boolean suspended = false;

  RunnableDemo( String name) {
    threadName = name;
    System.out.println("Creating " +  threadName );
  }

  public void run() {
    System.out.println("Running " +  threadName );
    try {
      for(int i = 10; i > 0; i--) {
        System.out.println("Thread: " + threadName + ", " + i);
        // Let the thread sleep for a while.
        Thread.sleep(300);
        synchronized(this) {
          while(suspended) {
            wait();
```

```
            }
          }
        }
      } catch (InterruptedException e) {
        System.out.println("Thread " +  threadName + " interrupted.");
      }
      System.out.println("Thread " +  threadName + " exiting.");
    }

    public void start () {
      System.out.println("Starting " +  threadName );
      if (t == null) {
        t = new Thread (this, threadName);
        t.start ();
      }
    }

    void suspend() {
      suspended = true;
    }

    synchronized void resume() {
      suspended = false;
      notify();
    }
}

public class TestThread {

    public static void main(String args[]) {

      RunnableDemo R1 = new RunnableDemo( "Thread-1");
      R1.start();

      RunnableDemo R2 = new RunnableDemo( "Thread-2");
      R2.start();

      try {
        Thread.sleep(1000);
        R1.suspend();
        System.out.println("Suspending First Thread");
        Thread.sleep(1000);
        R1.resume();
        System.out.println("Resuming First Thread");

        R2.suspend();
        System.out.println("Suspending thread Two");
        Thread.sleep(1000);
        R2.resume();
        System.out.println("Resuming thread Two");
      } catch (InterruptedException e) {
        System.out.println("Main thread Interrupted");
      }try {
        System.out.println("Waiting for threads to finish.");
        R1.t.join();
        R2.t.join();
```

```
      } catch (InterruptedException e) {
         System.out.println("Main thread Interrupted");
      }
      System.out.println("Main thread exiting.");
   }
}
```

The above program produces the following output −

Output

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 10
Running Thread-2
Thread: Thread-2, 10
Thread: Thread-1, 9
Thread: Thread-2, 9
Thread: Thread-1, 8
Thread: Thread-2, 8
Thread: Thread-1, 7
Thread: Thread-2, 7
Suspending First Thread
Thread: Thread-2, 6
Thread: Thread-2, 5
Thread: Thread-2, 4
Resuming First Thread
Suspending thread Two
Thread: Thread-1, 6
Thread: Thread-1, 5
Thread: Thread-1, 4
Thread: Thread-1, 3
Resuming thread Two
Thread: Thread-2, 3
Waiting for threads to finish.
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
Main thread exiting.
```

Java Networking

Java Networking is a concept of connecting two or more computing devices together so that we can share resources. Java socket programming provides facility to share data between different computing devices.

Advantage of Java Networking

1. sharing resources
2. centralize software management

Java Networking Terminology

The widely used java networking terminologies are given below:

1. IP Address
2. Protocol
3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol

6. Socket

1) IP Address

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255.

It is a logical address that can be changed.

2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

- TCP
- FTP
- Telnet
- SMTP
- POP etc.

3) Port Number

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

4) MAC Address

MAC (Media Access Control) Address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC.

5) Connection-oriented and connection-less protocol

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.

But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

6) Socket

A socket is an endpoint between two way communication.

Visit next page for java socket programming.

What we will learn in Networking Tutorial

- Networking and Networking Terminology
- Socket Programming (Connection-oriented)
- URL class
- Displaying data of a webpage by URLConnection class
- InetAddress class
- DatagramSocket and DatagramPacket (Connection-less)

Java Socket Programming

Java Socket programming is used for communication between the applications running on different JRE.

Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Important methods

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Important methods

Method	Description
1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

Example of Java Socket Programming

Let's see a simple of java socket programming in which client sends a text and server receives it.

File: MyServer.java

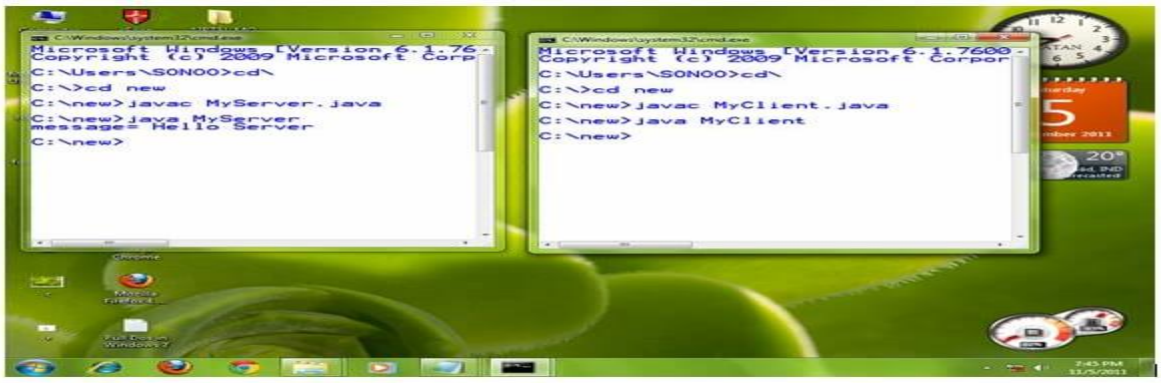
```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e){System.out.println(e);}  }  }
```

File: MyClient.java

```
import java.io.*;
import java.net.*;
public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}  }  }
```

To execute this program open two command prompts and execute each program at each command prompt as displayed in the below figure.

After running the client application, a message will be displayed on the server console.



Example of Java Socket Programming (Read-Write both side)

In this example, client will write first to the server then server will receive and print the text. Then server will write to the client and client will receive and print the text. The step goes on.

File: MyServer.java

```
import java.net.*;
import java.io.*;
class MyServer{
public static void main(String args[])throws Exception{
ServerSocket ss=new ServerSocket(3333);
Socket s=ss.accept();
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
  String str="",str2="";
while(!str.equals("stop")){
str=din.readUTF();
System.out.println("client says: "+str);
str2=br.readLine();
dout.writeUTF(str2);
dout.flush();
}
din.close();
s.close();
ss.close();
}}
```

File: MyClient.java

```
import java.net.*;
import java.io.*;
class MyClient{
public static void main(String args[])throws Exception{
Socket s=new Socket("localhost",3333);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
  String str="",str2="";
while(!str.equals("stop")){
str=br.readLine();
dout.writeUTF(str);
dout.flush();
str2=din.readUTF();
System.out.println("Server says: "+str2);
}
  dout.close();
s.close();
}}
```

Java URL

The **Java URL** class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web. For example:

1. http://www.javatpoint.com/java-tutorial

A URL contains many information:

1. **Protocol:** In this case, http is the protocol.
2. **Server name or IP Address:** In this case, www.javatpoint.com is the server name.
3. **Port Number:** It is an optional attribute. If we write http//ww.javatpoint.com:80/sonoojaiswal/ , 80 is the port number. If port number is not mentioned in the URL, it returns -1.
4. **File Name or directory name:** In this case, index.jsp is the file name.

Commonly used methods of Java URL class

The java.net.URL class provides many methods. The important methods of URL class are given below.

Method	Description
public String getProtocol()	it returns the protocol of the URL.
public String getHost()	it returns the host name of the URL.
public String getPort()	it returns the Port Number of the URL.
public String getFile()	it returns the file name of the URL.
public URLConnection openConnection()	it returns the instance of URLConnection i.e. associated with this URL.

Example of Java URL class

```
//URLDemo.java
import java.io.*;
import java.net.*;
public class URLDemo{
public static void main(String[] args){
try{
URL url=new URL("http://www.javatpoint.com/java-tutorial");

System.out.println("Protocol: "+url.getProtocol());
System.out.println("Host Name: "+url.getHost());
System.out.println("Port Number: "+url.getPort());
System.out.println("File Name: "+url.getFile());

}catch(Exception e){System.out.println(e);}
}
}
```

Output:

```
Protocol: http
Host Name: www.javatpoint.com
Port Number: -1
File Name: /java-tutorial
```

Java URLConnection class

The **Java URLConnection** class represents a communication link between the URL and the application. This class can be used to read and write data to the specified resource referred by the URL.

How to get the object of URLConnection class

The openConnection() method of URL class returns the object of URLConnection class. Syntax:

```
1. public URLConnection openConnection()throws IOException{}
```

Displaying source code of a webpage by URLConnecton class

The URLConnection class provides many methods, we can display all the data of a webpage by using the getInputStream() method. The getInputStream() method returns all the data of the specified URL in the stream that can be read and displayed.

Example of Java URLConnection class

```
import java.io.*;
import java.net.*;
public class URLConnectionExample {
public static void main(String[] args){
try{
URL url=new URL("http://www.javatpoint.com/java-tutorial");
```

```
URLConnection urlcon=url.openConnection();
InputStream stream=urlcon.getInputStream();
int i;
while((i=stream.read())!=-1){
System.out.print((char)i);
}
}catch(Exception e){System.out.println(e);}
}
}
```

Java HttpURLConnection class

The **Java HttpURLConnection** class is http specific URLConnection. It works for HTTP protocol only.

By the help of HttpURLConnection class, you can information of any HTTP URL such as header information, status code, response code etc.

The java.net.HttpURLConnection is subclass of URLConnection class.

How to get the object of HttpURLConnection class

The openConnection() method of URL class returns the object of URLConnection class. Syntax:

```
1. public URLConnection openConnection()throws IOException{}
```

You can typecast it to HttpURLConnection type as given below.

```
1. URL url=new URL("http://www.javatpoint.com/java-tutorial");
2. HttpURLConnection huc=(HttpURLConnection)url.openConnection();
```

Java HttpURLConnecton Example

```
import java.io.*;
import java.net.*;
public class HttpURLConnectionDemo{
public static void main(String[] args){
try{
URL url=new URL("http://www.javatpoint.com/java-tutorial");
HttpURLConnection huc=(HttpURLConnection)url.openConnection();
for(int i=1;i<=8;i++){
System.out.println(huc.getHeaderFieldKey(i)+" = "+huc.getHeaderField(i));
}
huc.disconnect();
}catch(Exception e){System.out.println(e);}
}
}
```

Output:

```
Date = Wed, 10 Dec 2014 19:31:14 GMT
Set-Cookie = JSESSIONID=D70B87DBB832820CACA5998C90939D48; Path=/
Content-Type = text/html
Cache-Control = max-age=2592000
Expires = Fri, 09 Jan 2015 19:31:14 GMT
Vary = Accept-Encoding,User-Agent
Connection = close
Transfer-Encoding = chunked
```

Java InetAddress class

Java InetAddress class represents an IP address. The java.net.InetAddress class provides methods to get the IP of any host name *for example*www.javatpoint.com, www.google.com, www.facebook.com etc.

Commonly used methods of InetAddress class

Method	Description
public static InetAddress getByName(String host) throws UnknownHostException	it returns the instance of InetAddress containing LocalHost IP and name.
public static InetAddress getLocalHost() throws UnknownHostException	it returns the instance of InetAdddress containing local host name and address.
public String getHostName()	it returns the host name of the IP address.
public String getHostAddress()	it returns the IP address in string format.

Example of Java InetAddress class

Let's see a simple example of InetAddress class to get ip address of www.javatpoint.com website.

```
import java.io.*;
import java.net.*;
public class InetDemo{
public static void main(String[] args){
try{
InetAddress ip=InetAddress.getByName("www.javatpoint.com");

System.out.println("Host Name: "+ip.getHostName());
System.out.println("IP Address: "+ip.getHostAddress());
}catch(Exception e){System.out.println(e);}
}
}
```

Output:

```
Host Name: www.javatpoint.com
IP Address: 206.51.231.148
```

Java DatagramSocket and DatagramPacket

Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

Java DatagramSocket class

Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets.

A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Commonly used Constructors of DatagramSocket class

- **DatagramSocket() throws SocketEeption:** it creates a datagram socket and binds it with the available Port Number on the localhost machine.
- **DatagramSocket(int port) throws SocketEeption:** it creates a datagram socket and binds it with the given Port Number.
- **DatagramSocket(int port, InetAddress address) throws SocketEeption:** it creates a datagram socket and binds it with the specified port number and host address.

Java DatagramPacket class

Java DatagramPacket is a message that can be sent or received. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Commonly used Constructors of DatagramPacket class

- **DatagramPacket(byte[] barr, int length):** it creates a datagram packet. This constructor is used to receive the packets.
- **DatagramPacket(byte[] barr, int length, InetAddress address, int port):** it creates a datagram packet. This constructor is used to send the packets.

Example of Sending DatagramPacket by DatagramSocket

```
//DSender.java
```

```
import java.net.*;
public class DSender{
  public static void main(String[] args) throws Exception {
    DatagramSocket ds = new DatagramSocket();
    String str = "Welcome java";
    InetAddress ip = InetAddress.getByName("127.0.0.1");

    DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ip, 3000);
    ds.send(dp);
    ds.close();
  }
}
```

Example of Receiving DatagramPacket by DatagramSocket

```
//DReceiver.java
import java.net.*;
public class DReceiver{
  public static void main(String[] args) throws Exception {
    DatagramSocket ds = new DatagramSocket(3000);
    byte[] buf = new byte[1024];
    DatagramPacket dp = new DatagramPacket(buf, 1024);
    ds.receive(dp);
    String str = new String(dp.getData(), 0, dp.getLength());
    System.out.println(str);
    ds.close();
  }
}
```