

Coordinate System

By Default, the upper left corner of a GUI component (such as applet or window) has the coordinates (0,0). A Coordinate pair is composed of x-coordinate (the horizontal coordinate) and a y-coordinate (the vertical coordinate). The x-coordinate is the horizontal distance moving right from the upper left corner.

The y-coordinate is the vertical distance moving down from the upper left corner. The x-axis describes every horizontal coordinate, and the y-axis describes every vertical coordinate.

Drawing Lines

call

```
g.drawLine(x1,y1,x2,y2)
```

method, where (x1, y1) and (x2, y2) are the endpoints of our lines and g is the Graphics object we are drawing with. The following program will result in a line on the applet.

```
import java.applet.*;
import java.awt.*;
public class SimpleLine extends Applet
{
    public void paint(Graphics g)
    {
        g.drawLine(10, 20, 30, 40);
    }
}
```

Drawing Rectangles

Drawing rectangles is simple. Start with a Graphics object g and call its drawRect() method:

```
public void drawRect(int x, int y, int width, int height)
g.drawRoundRect(10,100,80,50,10,10);
```

The first argument int is the left hand side of the rectangle, the second is the top of the rectangle, the third is the width and the fourth is the height.

Drawing Ovals and Circles

Java has methods to draw outlined and filled ovals. These methods are called drawOval(), and fillOval() respectively. These two methods are:

```
public void drawOval(int left, int top, int width, int height)
public void fillOval(int left, int top, int width, int height)
```

Instead of dimensions of the oval itself, the dimensions of the smallest rectangle, which can enclose the oval, are specified.

If the width and height are same oval becomes a circle.

```

public void paint(Graphics g)
{
    g.drawOval(20,20,200,120);
    g.setColor(Color.green);
    g.fillOval(70,30,100,100); // This is a circle.
}

```

Drawing Arcs

An arc is a part of an oval. The drawArc() designed to draw arcs takes six arguments. The first four are the same as the arguments for drawOval() method and the last two represent the starting angle of the arc and the number of degrees (sweep) angle around the arc.

```

g.drawArc(60,125,80,40,180,180);

import java.awt.*;
import java.applet.*;
public class line extends Applet
{
    public void paint(Graphics g)
    {
        g.drawArc(50,50,100,50,180,180);
        g.drawOval(60,125,200,200);
        g.drawOval(60,125,200,100);
        g.drawRect(10,60,40,30);
    }
}

```

USER INTERFACE COMPONENTS

The Button

To create a button, use one of the following constructors:

Button() creates a button with no text label.

Button(String) creates a button with the given string as label.

Example:

```

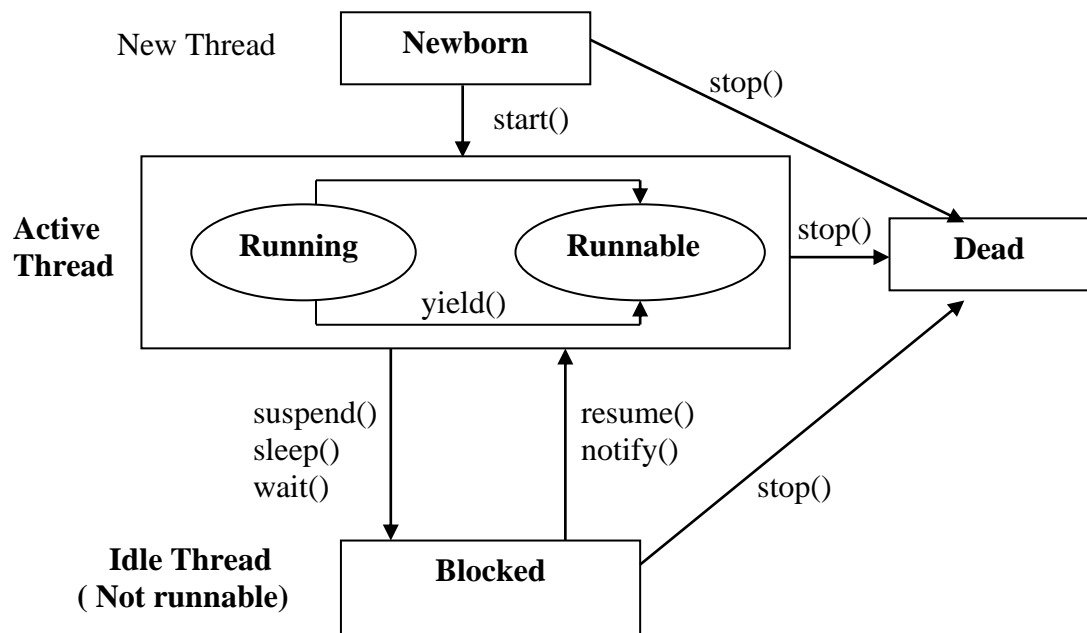
import java.awt.*;
import java.applet.Applet;
public class ButtonTest extends Applet
{
    Button b1 = new Button ("Play");
    Button b2 = new Button ("Stop");
    public void init()
    {
        add(b1);
        add(b2);
    }
}

```

Q) Explain the states of a thread with figure.

Ans :

Life Cycle of a thread :



1) newborn state -> The thread is not yet scheduled for running.

2) runnable state -> The runnable state means that the thread is ready for execution and is waiting for the availability of the processor.

3) Running state -> It means that the processor has given its time to thread for its execution.

4) Blocking a thread -> A thread can also be temporarily suspended or blocked.

a) sleep() -> blocked for a specified time.

b) suspend() -> blocked until further orders.

c) wait() -> blocked until certain condition occurs.

The thread will return to the runnable state when the specified time is elapsed in the case of sleep() and the resume() method is invoked in case of suspend() and the notify() method is called in the case of wait().

5) Stopping a thread -> A thread move to the dead state.

A thread will also move to the dead state automatically when it reaches the end of its method.

Example :

```
class A extends Thread
{
    public void run()
    {
```

```

        for(int i=1;i<=5;i++)
        {
            if(i==1) yield();
            System.out.println("i=" +i);
        }
        System.out.println("Exit from A");
    }
}
class B extends Thread
{
    public void run()
    {
        for(int j=1;j<=5;j++)
        {
            System.out.println("j=" +j);
            if(j==3) stop();
        }
        System.out.println("Exit from B");
    }
}
class C extends Thread
{
    public void run()
    {
        for(int k=1;k<=5;k++)
        {
            System.out.println("k=" +k);
            if(k==1)
            try
            {
                sleep(3000);
            }
            catch(Exception e)
            {}
        }
        System.out.println("Exit from C");
    }
}
class thread
{
    public static void main(String args[])
    {
        A a=new A();
        B b=new B();
        C c=new C();
        a.start();
        b.start();
        c.start();
    }
}

```

Thread priority

Each thread is assigned a priority, which affects the order in which it is scheduled for running.

Java permits us to set the priority of a thread using the setPriority() method.

Threadname.setPriority(int number);

MIN_PRIORITY=1;

NORM_PRIORITY=5;

MAX_PRIORITY=10;

```
class A extends Thread
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("i=" +i);
        }
        System.out.println("Exit from A");
    }
}
class B extends Thread
{
    public void run()
    {
        for(int j=1;j<=5;j++)
        {
            System.out.println("j=" +j);
        }
        System.out.println("Exit from B");
    }
}
class C extends Thread
{
    public void run()
    {
        for(int k=1;k<=5;k++)
        {
            System.out.println("k=" +k);
        }
        System.out.println("Exit from C");
    }
}
class threada
{
    public static void main(String args[])
    {
        A a=new A();
        B b=new B();
        C c=new C();
        a.setPriority(Thread.MIN_PRIORITY);
```

```

a.start();
b.setPriority(Thread.MAX_PRIORITY);
b.start();
c.setPriority(Thread.NORM_PRIORITY);
c.start();
}
}

```

Synchronization

When one thread may try to read a record from a file while another is still writing to the same file then it may get strange results. Java enables us to overcome this problem using a technique known as synchronization.

```

synchronized void update()
{
    // code here is synchronized
}

```

When we declare a method synchronized, java creates a “monitor” hands it over to the thread that calls the thread holds monitor, no other thread can enter the synchronized section of code.

Whenever a thread has completed its work of using synchronized method, it will hand over the monitor to the next thread that is ready to use the same resource.

Implementing the Runnable interface

We can also create thread using runnable interface.

- i) Declare the class as implementing the runnable interface.
- ii) implement the run() method.
- iii) create a thread by defining an object that is instantiated from this “runnable” class as the target of the thread.
- iv) call the threads start() method to run the thread.

```

class X implements Runnable
{
    public void run()
    {
        for(int i=1;i<=10;i++)
        {
            System.out.println("\tThread x : " + i);
        }
        System.out.println("end of threadX");
    }
}
class rtest
{
    public static void main(String args[])
    {
        X r=new X();
        Thread tx=new Thread(r);
    }
}

```

```

    tx.start();
    System.out.println("End of main thread");
}
}

```

Q) Explain the ‘extends’ keyword with an example.

Ans : ‘extends’ is a keyword where we can achieve the properties of inheritance.

Inheritance is the mechanism of obtaining the properties from one class to another class.

Note : The process of inheritance is also known as sub classing or extendable classes or reusability.

Example :

```

class student
{
    String name;
    int roll,age;
    void getdata()
    {
        name="Rohit";
        roll=5;
        age=25;
    }
    void putdata()
    {
        System.out.println("name=" +name);
        System.out.println("roll=" +roll);
        System.out.println("age=" +age);
    }
}
class marks extends student
{
    int m1,m2;
    void getm()
    {
        m1=75;
        m2=92;
    }
    void putm()
    {
        System.out.println("m1=" +m1);
        System.out.println("m2=" +m2);
    }
}
class sports extends marks
{
    int spwt,total;
    void putwt()

```

```

    {
        spwt=80;
        putdata();
        putm();
        System.out.println("spwt=" +spwt);
        total=m1+m2+spwt;
        System.out.println("Total=" +total);
    }
}
class inherit
{
    public static void main(String args[])
    {
        sports s = new sports();
        s.getdata();
        s.getm();
        s.putwt();
    }
}

```

Recursion

A method that calls itself is said to be recursive.

```

class fact
{
    int fact(int n)
    {
        int result;
        if(n==1) return 1;
        result=fact(n-1) *n;
        return result;
    }
}
class r
{
    public static void main(String args[])
    {
        fact f=new fact();
        System.out.println("Factorial of 3 is" + f.fact(3));
        System.out.println("Factorial of 4 is" +f.fact(4));
        System.out.println("Factorial of 5 is" +f.fact(5));
    }
}

```

Converting Strings to Numbers

When processing user input, it is often necessary to convert a String that the user enters into an int.

Syntax :

Integer.valueOf (Strings) and int Value () methods from the java.lang.Integer class. To convert the string “22” into the int .

```
int I = Integer.valueOf (“22”).intValue();
```

Doubles, floats and long are converted similarly. To convert a String like “22” into the long value 22, we would write

```
long1 = Long.valueOf(“22”).longValue();
```

To convert “22.5” into a float or a double, we would write:

```
double x = Double.valueOf(“22.5”).doubleValue();
```

```
float y = Float.valueOf(“22.5”).floatValue();
```

The various **valueOf()** methods are relatively intelligent and can handle plus and minus signs, exponents, and most other common number formats. However, if we pass one of these methods, something completely non-numeric, like “hello world,” it will throw a **NumberFormatException**.

```
class area
```

```
{  
public static void main (String args[])  
{  
double radius;  
double pi = 3.14;  
double A  
radius = Double.valueOf(args[0]).doubleValue ();  
A = pi*radius*radius;  
System.out.println(A + “square meters”)  
}  
}
```

WRAPPER CLASSES

In Java, primitive data types such as long, int, char etc. are not treated as objects. This is for simplicity and efficiency; but many times, we need to treat these primitive type as objects, for example, when we want to store primitive data types along with other objects in an array. The Java programming language provides “**Wrapper**” to manipulate primitive data elements as objects. Each primitive data type has a corresponding wrapper class in the java.lang package. These classes are:

Primitive Data Type	Wrapper class
Boolean	Boolean
Byte	Byte
Char	Character
Short	Short
Int	Integer
Long	Long
Float	Float