# Chapter 1

# JAVA Basics

Displaying results in Java

- System.out.println("message"+variable1+variable2+ "message"+…….);
  OR
  System.out.print("message"+variable1+variable2+ "message"+…….);
- Difference is println: display's & then go to new line, print: display's & stays on the same line
- JAVA is Case Sensitive
- system.out.println IS WRONG

Understand the object oriented nature of Java

- Java has 2 type of functions
  - Static
  - Non-static
- Functions (methods) within the class is normally accessed using
  object_name.function_name(arguments);
  (THIS IS VALID for NON-STATIC's)
- For STATIC functions it is
  class_name.function_name(arguments);

Header files

- While writing programs we will use the manufacturer available functions in the available classes
- Hence, we need to include the correct header files (called importing)
- NOTE that JAVA organizes functions in classes (as against C++ where the ready made functions were in header files) & a collection of classes into packages
- Thus we import (like #include) the specific class whose function we are using in our program OR we can import the package which contains the class whose function we are using.
- NOTE packages can also contain sub-packages & by importing a package it does NOT imply that the sub-package too is imported.

An Example

- E.g. java is one of the main packages which has many sub-packages like io, awt, lang, etc…. Each of these have many classes & each of these classes have many functions.
- E.g. java → lang → Math
  Here Math is a class in package lang which is within the package java
- Hence to use any function of class Math in our program we have to
  import java.lang.Math;
  OR
  import java.lang.*;
  doing a import java.*; will not import Math, since importing a package does NOT imply that the sub-package too is imported.
- What functions are there in Math class? Are they STATIC of not?

Methods of Math

- Math.sin(x)
- Math.cos(x)
- Math.tan(x)
- Math.asin(y)
- Math.acos(y)
- Math.atan(y)
- Math.pow(x,y)
- Math.exp(x)
- Math.log(x)
- Math.sqrt(x)
- Math.ceil(x)
- Math.floor(x)
- Math.round(x)
- Math.abs(x)
- Etc….

As seen, all the functions are called using

class_name.function_name(argument);

This is because all the functions are STATIC functions

When CAPITAL when SMALL?

- All class names start in capital letters.
- Every new sub-name also starts in capital e.g. a class called 'buffered reader' is written as BufferedReader
- All object name should (not necessary) start in capital
- Every function name start in SMALL letters e.g. println
- Every new function name component thereafter starts in capital e.g. a function called 'parse int' is written as parseInt

**Program to add 100.24 & 35.56**

```
import java.lang.*;
public class ex1
{
        public static void main(String args[ ])
        {
                double x=100.24,y=35.56,z;
                z=x+y;
                System.out.println("Sum="+z);
        }
}
```

- Import used since System class is within the java→lang package
  (NOTE: java.lang package is by default imported into every JAVA program, hence you need not import this from now on).
- Every main program i.e. main() has to be within a class.

- That class has to be public (so that the JVM: Java Virtual Machine, can enter it to execute the main(). Recollect the public access specifier meant – accessible by all)
- The program has to be saved as public_class_name.java
  In Current example we have to save the program as ex1.java
- Since every program has a main, every program will have one & only ONE public class & the filename for saving has to be that class name & .java extension
- main() function too has to be public, since JVM need to enter this function too (treat JVM like an outsider).
- 'static' word indicates just one instance, common to all, shared.
- Obviously, there can be only one main() in every program. Hence, main() has to be static.
- void means nothing. The return type of main() i.e. the type of data main returns, to its caller JVM, when it completes is NOTHING, i.e. main is not going to return any data. main() can return if it feel like & in that case it won't be void.
- We shall discuss about
  String args[ ]
  much later…..
  But, one thing for certain String has to be an inbuilt class (NOTE the capital letter start) & of the package java.lang
- Later we will study the functions (methods) of the class String

SUMMARY

- main within public class
- public class name is filename
- Only one public class in every program
- main has to be static
- Static fns called as
  class_name.function_name(…);
  e.g. y=Math.sqrt(4.3);
- Non-static functions called as
  object_name.function_name(…);
- Class names start in Capital & function names start in Small

Objects are like Pointers

- If we are going to access non-static functions then we have to create an object of that class
- The object we create is not exactly an object but a reference (pointer) to the object
- As we know memory to which the pointer points has to be created (dynamically), using new

e.g. If there is a class Employee
Complex X=new Complex();

- As seen above we create X dynamically using new.
- Recollect that the function who creates memory for the object is a constructor & it has the same name as the class. Hence, Complex() after new is the call to the default constructor
- If we had to use the parameterized constructor then it would be
  complex X=new complex(10,5);
- NOTE that now we don't have t do anything special for 'Copy Constructor' since in JAVA all objects are always references (Remember that we used to put the '&' in the parameter of the copy constructor, when we used to define it).
- Now we will never need that in case of copy constructors

```
complex(complex Y)
{
        ……….
}
```

- NOTE there is nothing in JAVA called pointer the way it was in C++. Means no more *'s and &'s.
- Thus the only way call be reference (change in formal parameters affects the actual parameters) can be achieved is by objects or arrays as arguments. Only these act as reference arguments, rest all are call by value(change in formal does not affect actual).

Inheritance in Java

- JAVA does not support
  - Multiple Inheritance
  - Virtual Base Class
  - Virtual Functions
  - Pure Virtual Functions.
- Setting up single inheritance is as follows

```
class Parent
{
        …….
}
class Child extends Parent
{
        …….
}
```

- Note extends is used by the child class to indicate it is an extension of which class.
- Where is the visibility mode???
- Will the inherited members come in private/public sections of the Child class???
- Just like C++ private does not inherit.
- public will inherit into public

- protected will inherit into protected
- NOTE no mode is NOT private mode, but it is like public mode in JAVA.
- protected is ACCESSIBLE from outside the class
- What's the difference in the various mode then?

| | Public | protected | No mode | private protected | private |
|---|---|---|---|---|---|
| Same class | Yes | Yes | Yes | Yes | Yes |
| Subclass in same package | Yes | Yes | Yes | Yes | No |
| Other class same package (outsider) | Yes | Yes | Yes | No | No |
| Subclass other packages | Yes | Yes | No | Yes | No |
| Non-subclass (outsider) other packages | Yes | No | No | No | No |

- See public, protected, no-mode columns closely

Constructors in Inheritance in Java

- Recollect in C++ child constructor had to call parent constructor in case of parameterized constructors & not in case of default constructors. Same for JAVA….

```
class B:private A
{       int y;
public:
        B(int a, int b):A(a)
        {
                y=b;
        }
}       //end of how its in C++
class B extends A
{
private: int y;
public:
        B(int a, int b)
        {
                super(a);           //calling parent constructor
                y=b;
        }
} //end of how its in JAVA
```

Scope of a variable in JAVA

- In C++, the following is valid

```
void main()
```

```
{       int x=10;
        cout<<x<<endl;
        {
                int x=5;
                cout<<x;
        }
}
```

Output:

10

5

- In JAVA, the following is an ERROR

```
class sample
{       public static void main(String args[ ])
        {
                int x=10;
        System.out.println(x);
                {
                        int x=5;
                System.out.println(x);
                }
        }
}
```

- Thus JAVA is not a block structured language with nested Scope of variables like C++.

SUMMARY

- JAVA does not support
  - Multiple Inheritance
  - Pointers in its original form
  - Virtual base class
  - Virtual functions
  - Pure virtual functions
  - Nested declaration of the same variable
  - Operator Overloading
  - Inline functions
  - Defining member functions outside the class & protyping them in the class
  - Friend functions & Friend classes
- JAVA supports
  - Control Statements (if, if-else, if-else-if, switch, while, do-while, for)
  - Arrays (Different from C++, here they are treated as Objects. More later…)
  - Function Overloading
  - Passing Object(s) as arguments
  - Returning Objects
  - Constructors
  - Inheritance (Not multiple inheritance)

**Reading Input**

- Java provides many ways of reading input from the users
- All data has to be read in the program into String variables, i.e. all data is assumed to be of String format

- Once read into String variables, they can be converted to other forms integer, double, float, etc…

Method 1

- The programmer creates a 'buffer' from which data is read line by line into String variable & converted if required.
- The buffer we create should be attached to the keyboard (System.in) through a pipe/channel (InputStreamReader).
- BufferedReader br=new BufferedReader(new InputStreamReader( System.in));
- 'br' is thus an object of the class BufferedReader. NOTE ever object is a reference, hence the 'new'.
- BufferedReader connected to InputStreamReader connected to System.in
- Next step is to read data through the buffer br.
  String str;
  str=br.readLine();
- NOTE that the LHS of readLine has to be a String (as mentioned earlier all data is accepted as String).
- NOTE readLine is a NON-STATIC function of BufferedReader class, hence an object was created of that class.
- The BufferedReader class falls under the io package of java, hence we need to import java.io.*;
- If the data wasn't a String but say an integer, then we will have to convert the String data to integer data in the program as follows:
- int x;
  x=Integer.parseInt(str);
- As can be seen, Integer is a class & parseInt is a STATIC function
- NOTE int is a datatype & Integer is a class
- Similarly, if you want to convert to double or float, the method is as follows
- double x;
  x=Double.parseDouble(str);
- float x;
  x=Float.parseFloat(str);
- Note that each datatype has a corresponding class associated with it. These classes are called Wrapper Classes. Obviously, the major role of these classes is to provide methods for data conversion.
- Problems can happen at two places
  - At readLine
  - At conversion (parsing)
- The first can be some sort of unexpected hardware error
- The second could be due to the user not entering the correct data
  x=Double.parseDouble(str)
  what if str is not containg a double value.
- There are two things which can be done
  - Either we handle the problem (Exception)
  - OR let the JVM handle the exception
- For the moment we let JVM handle the Exception & for this we need to tell it.
- The function in which the exception can occur needs to 'throw' it away so the JVM will catch it & handle it
  e.g.

```
public static void main(String args[ ]) throws IOException
{
        ……..
}
```

Method 2 (By Command Line Arguments)

- Here data will be entered at the command line & the arguments of main (which I have called args) will receive it.
- Being String arguments we can convert it (if required) using the parse functions.
- E.g.

```
public class ex
{
public static void main(String args[ ])          throws IOException
{       int a;
        String b;
        a=Integer.parseInt(args[0]);
        b=args[1];
        ……..
}
}
```

Method 3

- The util package provides a class called Scanner which has member functions to directly get the input in correct format.
- Scanner s=new Scanner(System.in);
  Here, s is an object of the scanner class & it is associated with the keyboard i.e. System.in
- The various data formats can be read as follows:

| | |
|---|---|
| To read an integer | x=s.nextInt(); |
| To read a long int | x=s.nextLong(); |
| To read a short int | x=s.nextShort(); |
| To read a float | x=s.nextFloat(); |
| To read a double | x=s.nextDouble(); |
| To read a boolean | x=s.nextBoolean(); |
| To read a Byte | x=s.nextByte(); |
| To read a String | x=s.next(); |
| To read a Line | x=s.readLine(); |

```
import java.util.Scanner;
public class scanner_example
{
        public static void main(String args[ ])
        {
                Scanner s=new Scanner(System.in);
                System.out.println("Feed in your age and birthdate");
                int a=s.nextInt();
                String str=s.next();
                System.out.println("Age="+ a);
                System.out.println("Birthdate="+str);
        }
}
```

**Operators, Expressions, Type Conversion and Precedence**

Arithmetic Operators:

- Integer Arithmetic

14 – 4 = 10 14 + 4 = 18 14 * 4 = 56
14 / 4 = 3 14 % 4 = 2 –14 % 3 = – 2
–14 % –3 = – 2 14 % –3 = 2

- Real Arithmetic

1.0 / 3.0 = 0.3333333333333333
–2.0 / 3.0 = – 0.666666666666666
NOTE: % operator CAN be used with real numbers (i.e. with float, double)
e.g. 20.5 % 6. 4 = 1.3

- Mixed – mode Arithmetic

15 / 10.0 = 1.5

Relational Operators

4.5 < = 5 True/False –10 < –11 True/False
20 > = 20 True/False 10 + 2 ! = 12 True/False
30 – 1 = = 29 True/False –35 > 10 True/False
NOTE that a relational operation evaluates to a 'true' or 'false'

Logical Operators

(IN JAVA 'true' is any number other than 0, 'false' is 0 IS INCORRECT)
AND &&
OR ||
e.g. a > b && d = = 10
The above statement is TRUE if both a>b and d=10
e.g. a > b || d = = 10
The above statement is TRUE if either a>b OR d=10 OR both
e.g. a > b || d = =10 && c < d
NOTE that AND has a higher precedence than OR
Thus the above example implies

a > b || (d = =10 && c < d)
(T/F) || ( T/F && T/F)
(Relational statements evaluate to a TRUE/FALSE)
e.g. a >b || true
The above statement is always TRUE
e.g. a > b && false
The above statement is always FALSE

Assignment Operators

- x = 5;
- x = x + y * 3;
  x = x + (y * 3);
  x + = y * 3;
- x = x + 1; x += 1;
- x = x – 3; x –= 3;
- x = x * 3; x *= 3;
- x = x / 3; x /= 3;
- x = x % y; x %= y;
- x = x * x; x *= x;

Advantages of Shorthand assignment operators:

- Whatever appears on the left need not be repeated & hence it becomes easier to write
- Statements become more concise & easier to read
- Statement becomes more efficient

Increment Decrement Operators

x = x + 1; x + +; + + x;
x = x – 1; x – –; – – x;
x = a++ + 5; x = a + 5; a = a + 1;
x = ++a + 5; a = a + 1; x = a + 5;

y = m + + is not same as y = + + m WHY?

Conditional Operator
More about it in the next chapter…..

**Objectives**

(1) y = 100 Find x and y after execution of

(a) x = y – –y * 4 Ans: x= 500; y=100
(b) x = y = y + + Ans: x=100; y=100
(c) x = y = = 100 Ans: Error Incompatible type
(d) x = y = = y + + Ans: Error Incompatible type

(2) Determine the value of the following expression:

–2 * –3 / 4 % 5 – –6 + 4 Ans: 11.0

(3) Determine the value of i , j, and k after execution

```
int i, j, k;
i = j = k = 1;
i –= –j – – – – –k;
```

Ans: i=2, j=0, k=0

(4) Determine the value of x, y, and z after execution

```
int x, z;
float y;
x = 5;
x /= y = z = 1 + 1.5;
```

[ABOVE IS AN ERROR 'possible loss of precision'. To not get an error explicit casting is a must

```
x /= y = z = (int)(1+1.5);
```

]

(5) Determine the output of the following code section

```
a=100;
System.out.print(a+ "~"+  + + a  +  "~" + a + +  +"~");
System.out.print(a);
```

OUTPUT:
100~101~101~102

**Programming Example 2:**
Read the principle, Rate and Time from user and calculate the Simple Interest & Compound Interest.

**Method1**

```
import java.io.*;
class SICI
{
   private double P,t,R,SI,CI;

   public void getdata() throws IOException
   {
      BufferedReader br=new BufferedReader(new
                                          InputStreamReader(System.in));
      System.out.println("Enter P,t,R");
      P=Double.parseDouble(br.readLine());
      t=Double.parseDouble(br.readLine());
      R=Double.parseDouble(br.readLine());
   }
   public void calculate()
   {
      SI=P*t*R/100;
      CI=P*Math.pow(1+R/100,t)-P;
   }

   public void putdata()
   {
      System.out.println("SI="+SI);
      System.out.println("CI="+CI);
   }
}
public class ex2_method1
{
    public static void main(String args[ ]) throws IOException
  {
      SICI X=new SICI();
      X.getdata();
      X.calculate();
      X.putdata();
  }
}
```

**Method 2**

```
import java.io.*;
public class ex2_method2
{
 public static void main(String args[ ]) throws IOException
 {
  double P,t,R,SI,CI;

  BufferedReader br=new BufferedReader(new
                                          InputStreamReader(System.in));

        System.out.println("Enter P,t,R");
        P=Double.parseDouble(br.readLine());
        t=Double.parseDouble(br.readLine());
        R=Double.parseDouble(br.readLine());

        SI=P*t*R/100;
        CI=P*Math.pow(1+R/100,t)-P;

        System.out.println("SI="+SI);
        System.out.println("CI="+CI);
 }
}
```

**Programming Example 3: PE3**
Write an interactive program that computes kmpl (km per litre) and cost per km for the operation of a vehicle based on the km traveled, gasoline consumed, cost of gasoline, and other operating costs

**Method 1**

```
import java.io.*;
class COST
{
  private double km,kmpl,gas,cgas,ocost,cpkm;
  public void getdata() throws IOException
  {
        BufferedReader br=new BufferedReader(new
                                              InputStreamReader(System.in));
        System.out.println("Feed in km traved, cost/l of gas,gas consumed, other
                                                                    costs");
        km=Double.parseDouble(br.readLine());
       cgas=Double.parseDouble(br.readLine());
        gas=Double.parseDouble(br.readLine());
       ocost=Double.parseDouble(br.readLine());
   }

  public void calculate()
  {
       kmpl=km/gas;
       cpkm=(gas*cgas+ocost)/km;
  }

  public void putdata()
  {
       System.out.println("km/l="+kmpl);
       System.out.println("cost/km="+cpkm);
  }
}
public class ex3_method1
{
 public static void main(String args[ ]) throws IOException
 {
      COST X=new COST();
      X.getdata();
      X.calculate();
      X.putdata();
 }
}
```

**Method 2**

```
import java.io.*;
public class ex3_method2
{
 public static void main(String args[ ]) throws IOException
 {
        double km,kmpl,gas,cgas,ocost,cpkm;

 BufferedReader br=new BufferedReader(new InputStreamReader (System.in));
   System.out.println("Feed in km travelled,cost/l of gas,gas consumed, other
                                                                    costs");
        km=Double.parseDouble(br.readLine());
        cgas=Double.parseDouble(br.readLine());
        gas=Double.parseDouble(br.readLine());
        ocost=Double.parseDouble(br.readLine());

        kmpl=km/gas;
        cpkm=(gas*cgas+ocost)/km;

        System.out.println("km/l="+kmpl);
        System.out.println("cost/km="+cpkm);
  }
}
```

**PE4:**
Write a program that reads the radius of a circle & determines its area, the area of the largest square contained within it, & the ratio of the two.

**Method 1**

```
import java.io.*;
class AREA
{
   private double r,AC,ASQ,Ratio;

   public void getdata() throws IOException
   {
      BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
      System.out.println("Enter radius of circle");
      r=Double.parseDouble(br.readLine());
   }

   public void calculate()
   {
       AC=Math.PI*r*r;
       ASQ=2*r*r;
       Ratio=AC/ASQ;
   }

   public void putdata()
   {
              System.out.println("Area of circle="+AC);
              System.out.println("Area of Largest Square="+ASQ);
              System.out.println("Ratio="+Ratio);
   }
}
public class ex4_method1
{
 public static void main(String args[ ]) throws IOException
 {
      AREA X=new AREA();
      X.getdata();
      X.calculate();
      X.putdata();
 }
}
```

**Method 2**

```
import java.io.*;
public class ex4_method2
{
 public static void main(String args[ ]) throws IOException
 {
        double r,AC,ASQ,Ratio;

  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter radius of circle");
        r=Double.parseDouble(br.readLine());

        AC=Math.PI*r*r;
        ASQ=2*r*r;
        Ratio=AC/ASQ;

        System.out.println("Area of circle="+AC);
        System.out.println("Area of Largest Square="+ASQ);
        System.out.println("Ratio="+Ratio);
   }
}
```

**PE5:** Write a program that determines the difference in the value of an investment after a given number of years when (i) the investment earns simple interest and (ii) the interest is compounded annually at the same rate.

**PE6:** A projectile fired at an angle θ with an initial velocity v travels a distance d given by $d = \frac{v^2}{g} \sin 2\theta$. It stays in motion for a time $t = \frac{2v}{g} \sin\theta$, Maximum height $h = \frac{v^2}{g} \sin\theta$
Write a program that computes d, t and h, given v and θ.

**Method 1**

```
import java.io.*;
class PROJECTILE
{
   private double v,g=9.81,th,d,t,h;

   public void getdata() throws IOException
   {
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Enter v,theta");
      v=Double.parseDouble(br.readLine());
      th=Double.parseDouble(br.readLine());
   }

   public void calculate()
   {
      d=v*v/g*Math.sin(2*th*Math.PI/180);
      t=2*v/g*Math.sin(th*Math.PI/180);
      h=v*v/g*Math.sin(th*Math.PI/180);
   }

   public void putdata()
   {  System.out.println("d="+d);
      System.out.println("t="+t);
      System.out.println("h="+h);
   }
}
public class ex6_method1
{
 public static void main(String args[ ]) throws IOException
 {
      PROJECTILE X=new PROJECTILE();
      X.getdata();
      X.calculate();
      X.putdata();
 }
}
```

**Method 2**

```
import java.io.*;
public class ex6_method2
{
 public static void main(String args[ ]) throws IOException
 {
        double v,g=9.81,th,d,t,h;

  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter v,theta");
        v=Double.parseDouble(br.readLine());
         th=Double.parseDouble(br.readLine());

        d=v*v/g*Math.sin(2*th*Math.PI/180);
        t=2*v/g*Math.sin(th*Math.PI/180);
        h=v*v/g*Math.sin(th*Math.PI/180);

        System.out.println("d="+d);
        System.out.println("t="+t);
        System.out.println("h="+h);
  }
}
```

**PE7:** Write a program that interchanges the value of x and y so that x has y 's value y has x's value.

**Method 1**

```
import java.io.*;
class SWAP
{
   private double x,y;

   public void getdata() throws IOException
   {
      BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
      System.out.println("enter 2 numbers");
      x=Double.parseDouble(br.readLine());
      y=Double.parseDouble(br.readLine());
   }

   public void calculate()
   {
      double t=x;
      x=y;
      y=t;
   }

   public void putdata()
   {
      System.out.println("New values="+x + " and "+y);
   }
}

public class ex7_method1
{
 public static void main(String args[ ]) throws IOException
 {
      SWAP X=new SWAP();
      X.getdata();
      X.calculate();
      X.putdata();
  }
}
```

**Method 2**

```
import java.io.*;
class SWAP
{
   private double x;

   public void getdata() throws IOException
   {
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      System.out.println("enter a numbers");
      x=Double.parseDouble(br.readLine());
   }

   public void calculate(SWAP Y)
   {
      double t=x;
      x=Y.x;
      Y.x=t;
   }

   public void putdata()
   {
      System.out.println("New values="+x);
   }
}

public class ex7_method2
{
 public static void main(String args[ ]) throws IOException
   {
      SWAP X=new SWAP();
      SWAP Y=new SWAP();
      X.getdata();
      Y.getdata();
      X.calculate(Y);
      X.putdata();
      Y.putdata();
   }
}
```

**Method 3**

```
import java.io.*;
public class ex7_method3
{
 public static void main(String args[ ]) throws IOException
 {
        double x,y;

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("enter 2 numbers");
        x=Double.parseDouble(br.readLine());
        y=Double.parseDouble(br.readLine());

        double t=x;
        x=y;
        y=t;

        System.out.println("New values="+x + " and "+y);
 }
}
```

**PE8:** Write a program to convert a given number of seconds into hours, minutes and seconds.

**Method 1**

```
import java.io.*;
class CONVERT
{
   private int h,m,s;

   public void getdata() throws IOException
   {
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Enter the seconds");
      s=Integer.parseInt(br.readLine());
   }

   public void calculate()
   {
        h=s/3600;
        s=s%3600;
        m=s/60;
        s=s%60;
   }

   public void putdata()
   {
     System.out.println("Converted time="+h+ " hr"+ m+" min "+s+" sec ");
   }
}

public class ex8_method1
{
 public static void main(String args[ ]) throws IOException
 {
       CONVERT X=new CONVERT();
       X.getdata();
       X.calculate();
       X.putdata();
 }
}
```

**Method 2**

```
import java.io.*;
public class ex8_method2
{
 public static void main(String args[ ]) throws IOException
 {
      int h,m,s;

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter the seconds");
        s=Integer.parseInt(br.readLine());

        h=s/3600;
        s=s%3600;
        m=s/60;
        s=s%60;

        System.out.println("Converted time="+h+ " hr"+ m+" min "+s+" sec ");
  }
}
```

**PE9:** Suppose that a particle gets into motion from rest & has constant acceleration 'a' for t seconds. The distance traveled 'd' by the particle & the final velocity 'v' are given by

$d = \frac{1}{2}at^2$ and $v = at$

Write a program that reads 'a' and 't', and prints 'd' and 'v'.

**PE10:** Write a program that reads three numbers, $x_1$, $x_2$, and $x_3$, and computes their average μ, their standard deviation σ, and the relative percentage RP for each numbers,

where $\mu = \frac{x1 + x2 + x3}{3}$

$\sigma^2 = \frac{x_1^2+x_2^2+x_3^2-3\mu^2}{3}$

$RP1 = \frac{x_1}{x_1+x_2+x_3}.100$ similarly RP2 & RP3

**Method 1**

```
import java.io.*;
class MSRP
{
   private double x1,x2,x3,m,s,rp1,rp2,rp3;

   public void getdata() throws IOException
   {
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Enter x1,x2,x3");
      x1=Double.parseDouble(br.readLine());
      x2=Double.parseDouble(br.readLine());
      x3=Double.parseDouble(br.readLine());
   }

   public void calculate()
   {
      m=(x1+x2+x3)/3;
      s=Math.sqrt((x1*x1+x2*x2+x3*x3-3*m*m)/3);
      rp1=x1/(x1+x2+x3)*100;
      rp2=x2/(x1+x2+x3)*100;
      rp3=x3/(x1+x2+x3)*100;
   }

    public void putdata()
   {
      System.out.println("Mean="+m);
      System.out.println("SD="+s);
      System.out.println("RP1="+rp1);
      System.out.println("RP2="+rp2);
      System.out.println("RP3="+rp3);
   }
}
```

```
public class ex10_method1
{
 public static void main(String args[ ]) throws IOException
 {
       MSRP X=new MSRP();
       X.getdata();
       X.calculate();
       X.putdata();
 }
}
```

**Method 2**

```
import java.io.*;
public class ex10_method2
{
 public static void main(String args[ ]) throws IOException
 {
       double x1,x2,x3,m,s,rp1,rp2,rp3;

  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Enter x1,x2,x3");
    x1=Double.parseDouble(br.readLine());
    x2=Double.parseDouble(br.readLine());
    x3=Double.parseDouble(br.readLine());

         m=(x1+x2+x3)/3;
         s=Math.sqrt((x1*x1+x2*x2+x3*x3-3*m*m)/3);
         rp1=x1/(x1+x2+x3)*100;
         rp2=x2/(x1+x2+x3)*100;
         rp3=x3/(x1+x2+x3)*100;

         System.out.println("Mean="+m);
         System.out.println("SD="+s);
         System.out.println("RP1="+rp1);
         System.out.println("RP2="+rp2);
         System.out.println("RP3="+rp3);
 }
}
```

**PE11:** The volume v of a sphere of radius r is given by
$v = (4/3)\ \pi r^3$. Write a program that computes the thickness of a hollow ball, given the total volume occupied by the ball and the volume inside the ball.

**Method 1**

```
import java.io.*;
class THICKNESS
{
   private double v,V,r,R,thick;

   public void getdata() throws IOException
   {
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Enter outer volume & inner volume");
      V=Double.parseDouble(br.readLine());
      v=Double.parseDouble(br.readLine());
   }

   public void calculate()
   {
      r=Math.pow(3*v/(4*Math.PI),1.0/3);
      R=Math.pow(3*V/(4*Math.PI),1.0/3);
      thick=R-r;
   }

   public void putdata()
   {
      System.out.println("Thickness="+thick);
   }
}

public class ex11_method1
{
 public static void main(String args[ ]) throws IOException
 {
      THICKNESS X=new THICKNESS();
      X.getdata();
      X.calculate();
      X.putdata();
 }
}
```

**Method 2**

```
import java.io.*;
public class ex11_method2
{
 public static void main(String args[ ]) throws IOException
 {
        double v,V,r,R,thick;

  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter outer volume & inner volume");
        V=Double.parseDouble(br.readLine());
        v=Double.parseDouble(br.readLine());

        r=Math.pow(3*v/(4*Math.PI),1.0/3);
        R=Math.pow(3*V/(4*Math.PI),1.0/3);
        thick=R-r;

        System.out.println("Thickness="+thick);
  }
}
```

**PE12:** The distance d between points $(x_1,y_1)$ and $(x_2,y_2)$ is given by $d=((x_1-x_2)^2+(y_1-y_2)^2)^{1/2}$ and the perimeter 'p' and the area 'a' of a triangle of sides of length u, v, and w are given by p=u+v+w and $a=\sqrt{(s(s-u)(s-v)(s-w))}$
where s=p/2. Write a program that reads the coordinates of three points of a triangle & prints its perimeter & area.

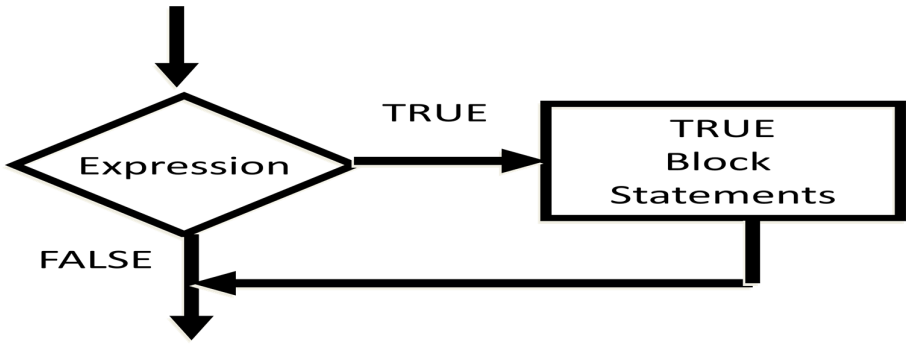
# Chapter 2

# Selective Programming

**if Statement**

```
if (expression)
{
        ……..
}
```

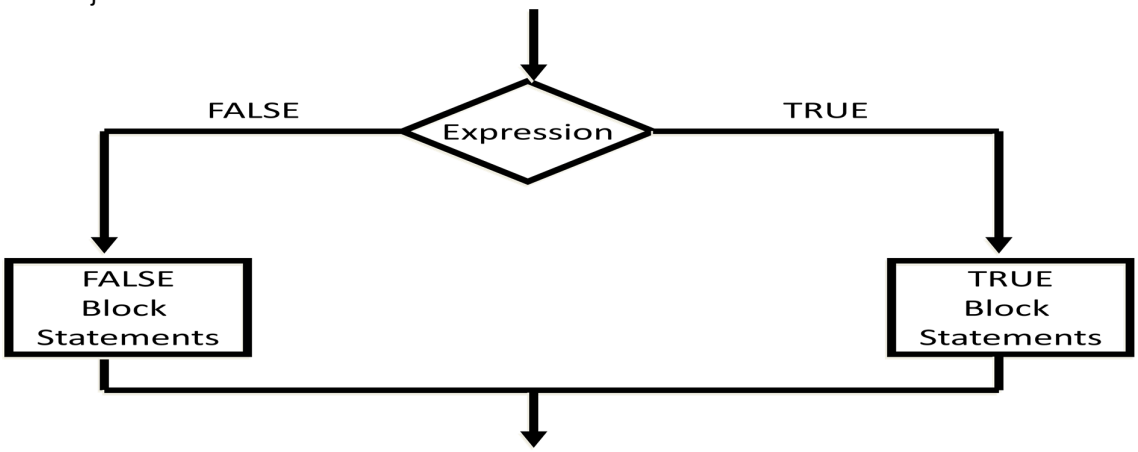
If the expression evaluates to a 'true' then the block of code is executed ELSE the block of code is skipped & the statements below it will yet be executed.




**PE14:** Write a program that finds the absolute value of a number.

**if – else Statement**

```
if (expression)
{
        ……../*TRUE block*/
}
else
{
        ……/*FALSE block*/
}
```




If the expression evaluates to a 'true' then the 'if' block of code is executed ELSE the 'if' block of code is skipped & 'else' block of code is executed. In either case the statements below it will yet be executed.

**PE15:** Write a program that finds the larger of two numbers.

**Method 1**

```
import java.io.*;
class Large
{
   private double a,b;

   public void getdata() throws IOException
   {
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Enter 2 numbers");
      a=Double.parseDouble(br.readLine());
      b=Double.parseDouble(br.readLine());
   }

   public void putdata()
   {
      if (a>b)
         System.out.println("Larger number is "+a);
      if (b>a)
         System.out.println("Larger number is "+b);
      if (a==b)
         System.out.println("Both are equal");
   }
}

public class ex15_method1
{
   public static void main(String args[ ]) throws IOException
   {
      Large X=new Large();
      X.getdata();
      X.putdata();
   }
}
```

**Method 2**

```
import java.io.*;
public class ex15_method2
{
 public static void main(String args[ ]) throws IOException
 {
       double a,b;

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter 2 numbers");
        a=Double.parseDouble(br.readLine());
        b=Double.parseDouble(br.readLine());

        if (a>b)
                System.out.println("Larger number is "+a);
        if (b>a)
                System.out.println("Larger number is "+b);
        if (a==b)
                System.out.println("Both are equal");
  }
}
```

**PE16:** Write a program to check whether a number is divisible by 3.

**Method 1**

```
import java.io.*;
class D3
{
   private int a;

   public void getdata() throws IOException
   {
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Enter a numbers");
      a=Integer.parseInt(br.readLine());
   }

  public void putdata()
  {
       if (a%3==0)
              System.out.println("Number is divisible by 3");
       else
              System.out.println("Number is NOT divisible by 3 ");
  }
}

public class ex16_method1
{
 public static void main(String args[ ]) throws IOException
 {
      D3 X=new D3();
      X.getdata();
      X.putdata();
  }
}
```

**Method 2**

```
import java.io.*;
public class ex16_method2
{
 public static void main(String args[ ]) throws IOException
 {
    int a;

  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Enter an integer");
    a=Integer.parseInt(br.readLine());

        if (a%3==0)
                System.out.println("Number is divisible by 3");
        else
                System.out.println("Number is NOT divisible by 3");
 }
}
```

**PE17:** Write a program to check whether a number is divisible by 3 or 7.

**Method 1**

```
import java.io.*;
class D3or7
{
   private int a;
   public void getdata() throws IOException
   {
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Enter a numbers");
      a=Integer.parseInt(br.readLine());
   }

   public void putdata()
   {
            if (a%3==0||a%7==0)
                  System.out.println("Number is divisible by 3 or 7");
            else
                  System.out.println("Number is NOT divisible by 3 and 7 ");
   }
}

public class ex17_method1
{
 public static void main(String args[ ]) throws IOException
 {
      D3or7 X=new D3or7();
      X.getdata();
      X.putdata();
 }
}
```

**Method 2**

```
import java.io.*;
public class ex17_method2
{
 public static void main(String args[ ]) throws IOException
 {
      int a;
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Enter an integer");
    a=Integer.parseInt(br.readLine());

       if (a%3==0 || a%7==0)
              System.out.println("Number is divisible by 3 or 7");
       else
              System.out.println("Number is NOT divisible by 3 and 7");
  }
}
```

**PE18:** Write a program to check whether a number is divisible by 3 and 7.

**Method 1**

```
import java.io.*;
class D3and7
{
   private int a;
   public void getdata() throws IOException
   {
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Enter a numbers");
      a=Integer.parseInt(br.readLine());
   }

   public void putdata()
   {
            if (a%3==0&&a%7==0)
                  System.out.println("Number divisible by 3 and 7");
            else
                  System.out.println("Number is NOT divisible by 3 or 7 ");
   }
}

public class ex18_method1
{
 public static void main(String args[ ]) throws IOException
 {
      D3and7 X=new D3and7();
      X.getdata();
      X.putdata();
 }
}
```

**Method 2**

```
import java.io.*;
public class ex18_method2
{
 public static void main(String args[ ]) throws IOException
 {
        int a;
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter an integer");
        a=Integer.parseInt(br.readLine());

        if (a%3==0 && a%7==0)
                System.out.println("Number divisible by 3 and 7");
        else
                System.out.println("Number is NOT divisible by 3 or 7");
   }
}
```

**PE19:** Write a program to check whether a number is even or odd.

**Method 1**

```
import java.io.*;
class EVENODD
{
   private int a;

   public void getdata() throws IOException
   {
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Enter a numbers");
      a=Integer.parseInt(br.readLine());
   }

   public void putdata()
   {
              if (a%2==0)
                     System.out.println("Number is EVEN");
              else
                     System.out.println("Number is ODD");
   }
}

public class ex19_method1
{
 public static void main(String args[ ]) throws IOException
 {
      EVENODD X=new EVENODD();
      X.getdata();
      X.putdata();
 }
}
```

**Method 2**

```
import java.io.*;
public class ex19_method2
{
 public static void main(String args[ ]) throws IOException
 {
       int a;
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
     System.out.println("Enter an integer");
     a=Integer.parseInt(br.readLine());

         if (a%2==0)
       System.out.println("Number is EVEN");
         else
       System.out.println("Number is ODD");
  }
}
```

**PE20:** Write a program to check whether a year is a leap year or not.

**Method 1**

```
import java.io.*;
class LEAP
{
   private int y;

   public void getdata() throws IOException
   {
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
     System.out.println("Enter a year");
     y=Integer.parseInt(br.readLine());
   }

   public void putdata()
   {
        if ((y%100!=0 && y%4==0)||y%400==0)
                System.out.println("Its a Leap Year");
        else
                System.out.println("Its NOT a Leap Year");
        }
}

public class ex20_method1
{
 public static void main(String args[ ]) throws IOException
 {
       LEAP X=new LEAP();
       X.getdata();
       X.putdata();
  }
}
```

**Method 2**

```
import java.io.*;
public class ex20_method2
{
 public static void main(String args[ ]) throws IOException
 {
        int y;
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter a year");
        y=Integer.parseInt(br.readLine());

        if ((y%100!=0 && y%4==0)||y%400==0)
                System.out.println("Its a LEAP Year");
        else
                 System.out.println("Its NOT a LEAP Year");
  }
}
```

**Conditional Operator (Ternary)**

```
if (expression)
{
        x = …(i)….;
}                                   ➔x =(expression)?(i) : (ii);
else
{
        x = …(ii)…;
}
```

**Nested Conditional Statements**

```
if (expression1)                    /*IF expression1 is TRUE*/
{
        if (expression2) {…….}      /*IF expression1 is TRUE and expression 2 is TRUE*/
        else {……..}                 /*IF expression1 is TRUE and expression 2 is FALSE*/
}
else                                /*IF expression1 is FALSE*/
{
        if (expression3) {……}       /*IF expression1 is FALSE and expression 3 is TRUE*/
        else {……}                   /*IF expression1 is FALSE and expression 3 is FALSE*/
}
```

**Dangling Else Problem**

```
if (expression1)
        if (expression2) {…….}
else
{
        if (expression3) {……}
        else {……}
}
```

Each ‘else’ is associated with the closest ‘if ’ (without an else). Hence the 1st ‘else’ is for expr2 & not for expr1.

To rectify it use braces.

```
if (expression1)
{
        if (expression2) {…….}          /*else–less if*/
}
else
{
        if (expression3) {……}
        else {……}
}
```

**Multiway Conditional Statement**

```
if (expression1)
        statement-1             /* If expression 1 is TRUE */
else if (expression2)
        statement-2             /* If expression 1 is FALSE and expression 2 is TRUE */
……..……
else
        statement-n             /* If ALL expressions are FALSE */
```

Only one statement block will be executed.

**PE21:**
Write a program to accept the 5 subject marks of a student & calculate the total & display the grade depending on percentage [60+ = A; 50 – 60=B; 40 – 50 =C; below 40=F]

**Method 1**

```
import java.io.*;
class GRADE
{   private int m1,m2,m3,m4,m5,total;
    private double avg;
    public void getdata() throws IOException
    {
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter the 5 subject marks");
        m1=Integer.parseInt(br.readLine());
        m2=Integer.parseInt(br.readLine());
        m3=Integer.parseInt(br.readLine());
        m4=Integer.parseInt(br.readLine());
        m5=Integer.parseInt(br.readLine());
    }

   public void calculate()
   {
       total=m1+m2+m3+m4+m5;
       avg=total/5.0;
   }

   public void putdata()
   {
       System.out.println("Total="+total);
       System.out.println("Percentage="+avg);
       if (avg>=60)
          System.out.println("Grade=A");
                else
          if(avg>=50)
             System.out.println("Grade=B");
          else
             if (avg>=40)
                System.out.println("Grade=C");
             else
                System.out.println("Grade=F");
    }
}
```

```
public class ex21_method1
{
 public static void main(String args[ ]) throws IOException
 {
       GRADE X=new GRADE();
       X.getdata();
       X.calculate();
       X.putdata();
  }
}
```

**Method 2**

```
import java.io.*;
public class ex21_method2
{
   public static void main(String args[ ]) throws IOException
   {
    int m1,m2,m3,m4,m5,total;
    double avg;

  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Enter the 5 subject marks");
    m1=Integer.parseInt(br.readLine());
    m2=Integer.parseInt(br.readLine());
    m3=Integer.parseInt(br.readLine());
    m4=Integer.parseInt(br.readLine());
    m5=Integer.parseInt(br.readLine());

        total=m1+m2+m3+m4+m5;
    avg=total/5.0;

    System.out.println("Total="+total);
    System.out.println("Percentage="+avg);

        if (avg>=60)  System.out.println("Grade=A");
      else if(avg>=50) System.out.println("Grade=B");
                       else if (avg>=40) System.out.println("Grade=C");
               else System.out.println("Grade=F");
   }
}
```

**Method 3**

```
import java.util.*;
class GRADE
{   private int m1,m2,m3,m4,m5,total;
    private double avg;

    public void getdata( )
    {
         Scanner s=new Scanner(System.in);
         System.out.println("Enter the 5 subject marks");
         m1=s.nextInt();
         m2=s.nextInt();
         m3=s.nextInt();
         m4=s.nextInt();
         m5=s.nextInt();
    }

   public void calculate()
   {
       total=m1+m2+m3+m4+m5;
       avg=total/5.0;
   }

   public void putdata()
   {
       System.out.println("Total="+total);
       System.out.println("Percentage="+avg);

       if (avg>=60)
          System.out.println("Grade=A");
                else
          if(avg>=50)
             System.out.println("Grade=B");
          else
             if (avg>=40)
                System.out.println("Grade=C");
             else
                System.out.println("Grade=F");
    }
}
```

```
public class ex21_method3
{
 public static void main(String args[ ]) throws IOException
 {
      GRADE X=new GRADE();
      X.getdata();
      X.calculate();
      X.putdata();
  }
}
```

**Method 4**

```
import java.util.*;
public class ex21_method4
{
   public static void main(String args[ ])
   {
      int m1,m2,m3,m4,m5,total;
      double avg;

      Scanner s=new Scanner(System.in);
      System.out.println("Enter the 5 subject marks");
      m1=s.nextInt();
      m2=s.nextInt();
      m3=s.nextInt();
      m4=s.nextInt();
      m5=s.nextInt();

      total=m1+m2+m3+m4+m5;
      avg=total/5.0;

      System.out.println("Total="+total);
      System.out.println("Percentage="+avg);

      if (avg>=60)System.out.println("Grade=A");
      else if(avg>=50) System.out.println("Grade=B");
                       else if (avg>=40) System.out.println("Grade=C");
                  else System.out.println("Grade=F");
   }
}
```

**Constant Multiway Conditional Statement**

```
switch(expression)
{
case value-1:  ……
               break;
case value-2:  ……
               break;
……
default:       ……
               break;
}
```

- Only one statement block will be executed, because of the break.
- Switch should not be used for a range of values.
- If distinct values have different actions to be performed then you can use switch.
- Case acts as an entry point & break acts like the exit point for the switch.
- The expression is executed & the value of the expression is matched with the case values to decide which case can be executed. If none satisfy then it chooses the default case.

**PE22:**

Write a program to simulate a basic arithmetic expression solver.
The user enters say
2.3+5.1
The result should be 7.400000
Similarly for - , * , / and ^.
General form
operand1 operator operand2

**PE23:**
Modify the above program to handle X and x as the alternative signs for *.

**PE24:**

A 1000-foot cable for a cable car is stretched between two towers, with a supporting tower midway between the two end towers. The velocity of the cable car depends on its position on the cable. When the cable car is within 30 ft of a tower, its velocity v is given in ft/sec as $v = 2.425 + 0.00175d^2$ where d is the distance in feet from the cable car to the nearest tower. If the cable car is not within 30ft of a tower, its velocity if given by $v = 0.625 + 0.12d – 0.00025d^2$

Write a program that reads the position of the cable car as the distance in feet from the 1st tower, and outputs the number of the nearest tower & velocity.

**PE25:** Write a program that reads the real coefficients a, b and c of the quadratic equation $ax^2+bx+c=0$ and computes the real roots.

**Method 1**

```
import java.util.Scanner;
class QUADRATIC
{
   private double a,b,c,root1,root2;
   public void getdata()
   {
      Scanner s=new Scanner(System.in);
      do
      {
         System.out.println("Enter coefficients a,b,c");
         a=s.nextDouble();
         b=s.nextDouble();
         c=s.nextDouble();
      }while(a==0 || b*b-4*a*c<0);
   }

   public void calculate()
   {
      root1=(-b+Math.sqrt(b*b-4*a*c))/(2*a);
      root2=(-b-Math.sqrt(b*b-4*a*c))/(2*a);
   }
   public void putdata()
   {
      System.out.println("Root1="+root1);
      System.out.println("Root2="+root2);
   }
}

public class ex25_method1
{
   public static void main(String args[ ])
   {
      QUADRATIC X=new QUADRATIC();
      X.getdata();
      X.calculate();
      X.putdata();
   }
}
```

**Method 2**

```
import java.util.*;
public class ex25_method2
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      double a,b,c,root1,root2;

       System.out.println("Enter coefficients a,b,c");
       a=s.nextDouble();
       b=s.nextDouble();
       c=s.nextDouble();

       if (a==0)
              System.out.println("Not a quadratic Equation");
       else if (b*b-4*a*c<0)
            System.out.println("Roots are imaginary");
       else
       {
                    root1=(-b+Math.sqrt(b*b-4*a*c))/(2*a);
                    root2=(-b-Math.sqrt(b*b-4*a*c))/(2*a);
                    System.out.println("Root1="+root1);
                    System.out.println("Root2="+root2);
       }
   }
}
```

**PE26:** Write a program that reads three positive numbers a,b,c and determines whether the triangle will be an obtuse angled, or a right angled, or an acute angled triangle. If the triangle is an acute angled triangle, determine further whether the triangle is equilateral, isosceles, or scalene.

**Method 1**

```
import java.util.*;
public class ex26
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      double a,b,c;
      do
      {
         System.out.println("Enter the 3 sides");
         a=s.nextDouble();
         b=s.nextDouble();
         c=s.nextDouble();
      }while(a<=0||b<=0||c<=0||a+b<=c||b+c<=a||a+c<=b);

       if(a*a>b*b+c*c || b*b>a*a+c*c || c*c>a*a+b*b)
              System.out.println("Triangle is obtuse");
       else if(a*a==b*b+c*c || b*b==a*a+c*c || c*c==a*a+b*b)
              System.out.println("Triangle is rt. angled");
       else
       {
              System.out.println("triangle is acute");
              if (a==b && b==c)
                     System.out.println("Triangle is equilateral");
              else if (a==b|| b==c || a==c)
                     System.out.println("Triangle is isosceles");
              else
                     System.out.println("Triangle is scalene");
        }
   }
}
```

**PE27:** Write a temperature conversion program. The program should read the temperature measured and the type of scale used in the measurement (F for Fahrenheit and C for Celsius) and convert it to the other scale.

**Method 1**

```
import java.util.*;
class CONVERT
{
   private double input,output;
   private String type;

   public void getdata()
   {
      Scanner s=new Scanner(System.in);
      System.out.println("Enter the temp & then type");
      input=s.nextDouble();
      type=s.next();
   }
   public void calculate( )
   {
        if (type.equals("F"))
          output=5*(input-32)/9;
        else
           output=9*input/5+32;
   }
   public void putdata()
   {
      System.out.print("Result="+output);
      if (type.equals("F"))    System.out.println(" C");
      else
         System.out.println(" F");
   }
}

public class ex27_method1
{
   public static void main(String args[ ])
   {
      CONVERT X=new CONVERT();
      X.getdata();
      X.calculate();
      X.putdata();
   }
}
```

**Method 2**

```
import java.util.*;
public class ex27_method2
{
   public static void main(String args[ ])
   {
      double input,output;
      String type;

      Scanner s=new Scanner(System.in);
      System.out.println("Enter the temp & then type");
      input=s.nextDouble();
      type=s.next();

       if (type.equals("F"))
         output=5*(input-32)/9;
      else
         output=9*input/5+32;

      System.out.print("Result="+output);
      if (type.equals("F"))
         System.out.println(" C");
      else
         System.out.println(" F");
   }
}
```

**P28:** A function f is defined as follows:
$f(x)=ax^3 – bx^2 + cx –d,$ if $x > k$
$f(x)=0$ if $x = k$
$f(x)= –ax^3+bx^2-cx+d,$ if $x < k$
Write a program that reads a, b, c, d, k, and x, and prints the value of f(x).

**Method 1**

```
import java.util.Scanner;
class FUNCTION
{
   private double a,b,c,d,x,k,fun;

   public void getdata()
   {
        Scanner s=new Scanner(System.in);
        System.out.println("Enter a,b,c,d,x,k");
        a=s.nextDouble();
        b=s.nextDouble();
        c=s.nextDouble();
        d=s.nextDouble();
        x=s.nextDouble();
      k=s.nextDouble();
   }

   public void calculate()
   {
      if (x>k)
         fun=a*Math.pow(x,3)-b*x*x+c*x-d;
      if (x==k)
         fun=0;
      if(x<k)
         fun=-a*Math.pow(x,3)+b*x*x-c*x+d;
   }

   public void putdata()
   {
      System.out.println("f(x)="+fun);
   }
}
```

```
public class ex28_method1
{
   public static void main(String args[ ])
   {
      FUNCTION X=new FUNCTION();
      X.getdata();
      X.calculate();
      X.putdata();
   }
}
```

**Method 2**

```
import java.util.*;
public class ex28_method2
{
   public static void main(String args[ ])
   {
      double a,b,c,d,x,k,fun=0;
               //have to initialize 'fun' since the assisnment is in if's & if's can
        fail
       //initialization is automatic in 2 class method
           //not reqd in if – else if - else
      Scanner s=new Scanner(System.in);
      System.out.println("Enter a,b,c,d,x,k");
      a=s.nextDouble();
      b=s.nextDouble();
      c=s.nextDouble();
      d=s.nextDouble();
      x=s.nextDouble();
      k=s.nextDouble();
      if (x>k)
         fun=a*Math.pow(x,3)-b*x*x+c*x-d;
      if (x==k)
         fun=0;
      if(x<k)
         fun=-a*Math.pow(x,3)+b*x*x-c*x+d;

      System.out.println("f(x)="+fun);
   }
}
```

**P29**: The commission on a salesman's total sales is as follows:
☺Sales<100 => No commission
☺100 ≤ Sales <1000 => Commission = 10% of the sales value
☺Sales ≥ 1000 => Commission = 100 + 12% of sales above 1000.
Write a program that reads sales and prints out the salesman's commission.

**Method 1**

```
import java.util.*;
class SALESMAN
{
   private double sales,commission;

   public void getdata()
   {
      Scanner s=new Scanner(System.in);
      System.out.println("Feed in the sales");
      sales=s.nextDouble();
   }
   public void calculate()
   {
        if (sales<100)
              commission=0;
        if (sales>=100 && sales<1000)
              commission=0.1*sales;
        if (sales>=1000)
              commission=100+0.12*(sales-1000);
   }

   public void putdata()
   {
        System.out.println("Commission="+commission);
   }
}
public class ex29_method1
{
   public static void main(String args[ ])
   {
      SALESMAN X=new SALESMAN();
      X.getdata();
      X.calculate();
      X.putdata();
   }
}
```

**Method 2**

```
import java.util.*;
public class ex29_method2
{
   public static void main(String args[ ])
   {
      double sales,commission=0;
               //necessary to initialize as value set in if's & if's can fail
       //not required in 2 class method & not reqd in if – else if - else
      Scanner s=new Scanner(System.in);
      System.out.println("Feed in the sales");
      sales=s.nextDouble();

       if (sales<100)
         commission=0;
       if (sales>=100 && sales<1000)
         commission=0.01*sales;
       if (sales>=1000)
         commission=100+0.12*(sales-1000);

       System.out.println("Commission="+commission);
   }
}
```

# Chapter 3

# Repetitive Programming

**WHILE LOOP**

```
while (logical condition)
{
        body of loop
}
statement-y;
```

The logical condition is evaluated & on success the loop body is executed & the condition is checked again. On success the loop body is executed again & again the loop condition is checked….. and so on… till this logical condition fails. Once the logical condition fails the loop body is skipped and program execution continues from statement-y.
If the logical condition had failed the very first time it was checked then the program would have directly jumped to statement-y.

**Loop interruption**

Loop execution can be interrupted using either break or continue.
'break' takes the control out of the loop & 'continue' take the control back to the start of the loop

```
        while (logical condition)
        {
                ---------
                break;
                ---------
        }

        statement-y;
        while (logical condition)
        {
                ---------
                continue;
                ---------
        }
        statement-y;
```

**DO – WHILE LOOP**

```
do
{
        body of loop
} while (logical condition);
statement-y;
```

The body of the loop is executed first, then the logical condition is checked. On success the control transfers to the 'do' and then the Loop body is executed again & again the condition is checked & if its TRUE again then again the control transfers to the 'do' and so on….. This stops when the logical condition fails. When it fails the control transfers to statement-y.
The rules of 'break' and 'continue' are the same for do-while as they were for while.

**FOR LOOP**

Another form of loop structure is the for loop.

It consists of 3 parts

- The initialization section (executed 1st and only once)
- The condition
- The step

```
for (initialization ; logical condition ; step)
{
    loop body
}
    statement-y;
```

```
for (initialization ; logical condition ; step)
{
        loop body
}
        statement-y;
```

*First the initialization takes place*

```
for (initialization ; logical condition ; step)
{
        loop body
}
        statement-y;
```

*Next, the logical condition is checked for TRUE/FALSE*

```
for (initialization ; logical condition ; step)
{
        loop body
}
        statement-y;
```

*If TRUE then the loop body is executed*

```
for (initialization ; logical condition ; step)
{
        loop body
}
        statement-y;
```

*After the loop body is executed the step comes into play, which is usually an increment/decrement.*

```
for (initialization ; logical condition ; step)
{
        loop body
}
        statement-y;
```

*Next, the logical condition is check again for TRUE/FALSE. If TRUE the loop body is executed again, then the step & the logical check once again.*

The procedure repeats again and again. This cycle (loop) stops when the logical condition fails. Then the program control transfers to the statement-y.

**Interruption of for loop**

The for loop can be interrupted the same way as while & do-while i.e. using the break or continue. The break will take us out of the for loop (to statement-y) and the continue will take us to the step & then the logical condition & then depending on the TRUE/FALSE value the loop body will be executed or the control will transfer to statement-y.

**PE30:** Print all integers from 1 to 100

```
public class ex30
{
   public static void main(String args[ ])
   {
      int i;
      for(i=1;i<=100;i++)
      {
         System.out.println(i);
      }
   }
}
```

**PE31:** Print the 1st N natural numbers

**Method 1 (using BufferedReader)**

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
public class ex31_method1
{
  public static void main(String args[ ]) throws IOException
 {
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        String str;
        int i,N;

        System.out.println("Feed in value of N:");
        str=br.readLine();
        N=Integer.parseInt(str);

        for(i=1;i<=N;i++)
                System.out.println(i);
  }
}
```

**Method 1 (using Scanner)**

```
import java.util.Scanner;
public class ex31_method2
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      int i,N;

      System.out.println("Feed in value of N:");
      N=s.nextInt();

       for(i=1;i<=N;i++)
                System.out.println(i);
   }
}
```

**PE32:** Print even numbers from 1 to N

**Method 1 (using BufferedReader)**

```
import java.io.*;
public class ex32_method1
{
 public static void main(String args[ ]) throws IOException
 {
        BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
        String str;
        int i,N;

        System.out.println("Feed in value of N:");
        str=br.readLine();
        N=Integer.parseInt(str);

        for(i=2;i<=N;i=i+2)
                System.out.println(i);
}
}
```

**Method 1 (using Scanner)**

```
import java.util.Scanner;
public class ex32_method2
{
   public static void main(String args[ ])
   {
                Scanner s=new Scanner(System.in);
                int i,N;
                System.out.println("Feed in value of N:");
                N=s.nextInt();
                for(i=2;i<=N;i=i+2)
                        System.out.println(i);
   }
}
```

**Method 2 (using BufferedReader)**

```
import java.io.*;
public class ex32_method3
{
   public static void main(String args[ ]) throws IOException
   {
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        String str;
        int i,N;

        System.out.println("Feed in value of N:");
        str=br.readLine();
        N=Integer.parseInt(str);

        for(i=1;i<=N;i++)
        {
                if (i%2==0)
                        System.out.println(i);
        }
   }
}
```

**Method 2 (using Scanner)**

```
import java.util.Scanner;
public class ex32_method4
{
    public static void main(String args[ ])
    {
                Scanner s=new Scanner(System.in);
                int i,N;

                System.out.println("Feed in value of N:");
                N=s.nextInt();

                for(i=1;i<=N;i++)
                        if(i%2==0)  System.out.println(i);
    }
}
```

**PE33:** Print all odd numbers from 1 to N

**Method 1 (BufferedReader)**

```
import java.io.*;
public class ex33_method1
{
 public static void main(String args[ ]) throws IOException
 {
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        String str;
        int i,N;

        System.out.println("Feed in value of N:");
        str=br.readLine();
        N=Integer.parseInt(str);

        for(i=1;i<=N;i=i+2)
                System.out.println(i);
  }
}
```

**Method 1 (Scanner)**

```
import java.util.Scanner;
public class ex33_method2
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      int i,N;

      System.out.println("Feed in value of N:");
      N=s.nextInt();

       for(i=1;i<=N;i=i+2)
               System.out.println(i);
        }
}
```

**Method 2 (BufferedReader)**

```
import java.io.*;
public class ex33_method3
{
 public static void main(String args[ ]) throws IOException
 {
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        String str;
        int i,N;

        System.out.println("Feed in value of N:");
        str=br.readLine();
        N=Integer.parseInt(str);

        for(i=1;i<=N;i++)
        {
                if (i%2!=0)
                        System.out.println(i);
        }
 }
}
```

**Method 2 (Scanner)**

```
import java.util.Scanner;
public class ex33_method4
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      int i,N;

        System.out.println("Feed in value of N:");
        N=s.nextInt();

        for(i=1;i<=N;i++)
        {
          if(i%2!=0)
                        System.out.println(i);
        }
   }
}
```

**PE34:** Print all multiples of 3 from 1 to N.

**Method 1**

```
import java.util.Scanner;
public class ex34_method1
{
    public static void main(String args[ ])
    {
       Scanner s=new Scanner(System.in);
       int N;
       System.out.println("Feed in the value of N:");
       N=s.nextInt();

      for(int i=3;i<=N;i=i+3)
            System.out.println(i);
      }
}
```

**Method 2**

```
import java.util.Scanner;
public class ex34_method2
{
    public static void main(String args[ ])
    {
       Scanner s=new Scanner(System.in);
       int N;
       System.out.println("Feed in the value of N:");
       N=s.nextInt();

       for(int i=1;i<=N;i++)
       {
            if(i%3==0)
                 System.out.println(i);
       }
    }
}
```

**PE35:** Print all multiples of 3 and 7 from 1 to N

**Method 1**

```
import java.util.Scanner;
public class ex35_method1
{
    public static void main(String args[ ])
    {
       Scanner s=new Scanner(System.in);
       int N;

       System.out.println("Feed in the value of N:");
       N=s.nextInt();

       for(int i=21;i<=N;i=i+21)
              System.out.println(i);
       }
}
```

**Method 2**

```
import java.util.Scanner;
public class ex35_method2
{
    public static void main(String args[ ])
    {
       Scanner s=new Scanner(System.in);
       int N;

       System.out.println("Feed in the value of N:");
       N=s.nextInt();
       for(int i=1;i<=N;i++)
       {
          if(i%3==0 && i%7==0)
              System.out.println(i);
       }
    }
}
```

**PE36:** Print all multiples of 3 or 7 from 1 to N

```
import java.util.Scanner;
public class ex36_method1
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      int N;

      System.out.println("Feed in the value of N:");
      N=s.nextInt();

      for(int i=1;i<=N;i++)
      {
        if(i%3==0 || i%7==0)
              System.out.println(i);
      }
   }
}
```

**PE37:** Print the sum of N numbers

**Method 1**

```
import java.util.Scanner;
public class ex37_method1
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      int N;
      double x,sum=0;

      System.out.println("Feed in the value of N:");
      N=s.nextInt();

      for(int i=1;i<=N;i++)
      {
        System.out.println("Feed in the next number ");
         x=s.nextDouble();
         sum=sum+x;
      }
      System.out.println("Sum="+sum);
   }
}
```

**Method 2**

```
import java.util.Scanner;
class SUM
{
        private double x[ ],sum;
        private int N;

        public void getdata()
        {
                Scanner s=new Scanner(System.in);
                System.out.println("Feed in N : ");
                N=s.nextInt();
                x=new double[N];

                System.out.println("Feed in the elements");
                for(int i=0;i<N;i++)
                        x[i]=s.nextDouble();
        }

        public void calculate()
        {
              int i;
              sum=0;
              for(i=0;i<N;i++)
                sum=sum+x[i];
        }

        public void putdata()
        {
                System.out.println("Sum="+sum);
        }
}

public class ex37_method2
{
        public static void main(String args[ ])
        {
                SUM X=new SUM();
                X.getdata();
                X.calculate();
                X.putdata();
        }
}
```

**PE38:** Find the factorial of a number.

**Method 1**

```
import java.util.Scanner;
public class ex38_method1
{
   public static void main(String args[ ])
   {
      int N;
      Scanner s=new Scanner(System.in);

      System.out.println("feed in N: ");
      N=s.nextInt();

       double p=1;
       for(int i=1 ;i<=N;i++)
       {
              p=p*i;
       }
      System.out.println("Factorial of "+N+" is "+p);
   }
}
```

**Method 2**

```
import java.util.Scanner;
class FACT
{
   private int N;
   private double p;

   public void getdata()
   {
      Scanner s=new Scanner(System.in);
      System.out.println("Feed in the value of N: ");
      N=s.nextInt();
   }

   public void calculate()
   {
      p=1;
      for(int i=1;i<=N;i++)
      {
         p=p*i;
      }
   }

   public void putdata()
   {
      System.out.println("Factorial of "+N+" is "+p);
   }
}

public class ex38_method2
{
   public static void main(String args[ ])
   {
      FACT X=new FACT();
      X.getdata();
      X.calculate();
      X.putdata();
   }
}
```

**PE39:**Print the multiplication table of 13. The output should be displayed as:
13 X 1 = _____
13 X 2 = _____
....
13 X 12=_____

```
public class ex39_method1
{
   public static void main(String args[ ])
   {
      int i;
      for(i=1;i<=12;i++)
      {
         System.out.println("13X"+i+"="+13*i);
      }
   }
}
```

**PE40:** Print the multiplication table of 13 from 13 X 1 to 13 X N.

```
import java.util.Scanner;
public class ex40_method1
{
   public static void main(String args[ ])
   {
      int i,N;
      Scanner s=new Scanner(System.in);

       System.out.println("Feed in N: ");
       N=s.nextInt();

       for(i=1;i<=N;i++)
                    System.out.println("13X"+i+"="+13*i);
       }
}
```

**PE41:** Print the multiplication table from P X M to P X N

**Method 1**

```
import java.util.Scanner;
public class ex41_method1
{
   public static void main(String args[ ])
   {
      int i,P,M,N;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in P,M,N: ");
      P=s.nextInt();
      M=s.nextInt();
      N=s.nextInt();

       if (M<N)
       {
              for(i=M;i<=N;i++)
              {
                     System.out.println(P+"X"+i+"="+P*i);
              }
       }
       else
       {
              for(i=M;i>=N;i--)
              {
                     System.out.println(P+"X"+i+"="+P*i);
              }
       }
   }
}
```

**Method 2**

```
import java.util.Scanner;
class Multi
{
   private int P,M,N;
   public void getdata()
   {
      Scanner s=new Scanner(System.in);
      System.out.println("Feed in P,M,N: ");
      P=s.nextInt();
      M=s.nextInt();
      N=s.nextInt();
   }

   public void putdata( )
   {
        int i;
        if (M<N)
        {
         for(i=M;i<=N;i++)
                System.out.println(P+"X"+i+"="+P*i);
        }
        else
        {
         for(i=M;i>=N;i--)
                 System.out.println(P+"X"+i+"="+P*i);
        }
   }
}
public class ex41_method2
{
   public static void main(String args[ ])
   {  Multi X=new Multi();
      X.getdata();
      X.putdata();
   }
}
```

**PE42:**Print the sum of the following series
(i) 1+2+3+….. N terms
(ii) 1+3+5+….. N terms
(iii) 2+4+6+….. N terms
(iv) 1+4+9+16+…. N terms
(v) 1+8+27+….N terms
(vi) 1+4+16+64+….. N terms
(vii) 1*3+2*6+3*9+….. N terms
(viii) 1+3+5+….. N terms
2+4+6+…. N terms
(ix) 1*3+2*6+3*9+…. N terms
1*4+3*8+5*12+… M terms

(i)

```
import java.util.*;
public class ex42_i_method1
{
   public static void main(String args[ ])
   {
      int sum,term,N;
      Scanner s=new Scanner(System.in);

      System.out.println("feed in N: ");
      N=s.nextInt();

      sum=0;
      term=1;
      for(int i=1;i<=N;i++)
      {
         sum=sum+term;
         term=term+1;
      }
      System.out.println("Sum="+sum);
   }
}
```

(ii)

```
import java.util.*;
public class ex42_ii_method1
{
   public static void main(String args[ ])
   {
      int sum,term,N;
      Scanner s=new Scanner(System.in);

      System.out.println("feed in N: ");
      N=s.nextInt();

      sum=0;
      term=1;
      for(int i=1;i<=N;i++)
      {
         sum=sum+term;
         term=term+2;
      }
      System.out.println("Sum="+sum);
   }
}
```

(iii)

```
import java.util.*;
public class ex42_iii_method1
{
   public static void main(String args[ ])
   {
      int sum,term,N;
      Scanner s=new Scanner(System.in);

      System.out.println("feed in N: ");
      N=s.nextInt();

         sum=0;
      term=2;
      for(int i=1;i<=N;i++)
      {
         sum=sum+term;
         term=term+2;
      }
      System.out.println("Sum="+sum);
   }
}
```

(iv)

```
import java.util.*;
public class ex42_iv_method1
{
   public static void main(String args[ ])
   {
      int sum,term,N;
      Scanner s=new Scanner(System.in);

      System.out.println("feed in N: ");
      N=s.nextInt();

      sum=0;
      term=1;
      for(int i=1;i<=N;i++)
      {
         sum=sum+term*term;
         term=term+1;
      }
      System.out.println("Sum="+sum);
   }
}
```

(v)

```
import java.util.*;
public class ex42_v_method1
{
   public static void main(String args[ ])
   {
      int sum,term,N;
      Scanner s=new Scanner(System.in);

      System.out.println("feed in N: ");
      N=s.nextInt();

      sum=0;
      term=1;
      for(int i=1;i<=N;i++)
      {
         sum=sum+term*term*term;
         term=term+1;
      }
      System.out.println("Sum="+sum);
   }
}
```

(vi)

```
import java.util.*;
public class ex42_vi_method1
{
   public static void main(String args[ ])
   {
      int sum,term,N;
      Scanner s=new Scanner(System.in);

      System.out.println("feed in N: ");
      N=s.nextInt();

      sum=0;
      term=1;
      for(int i=1;i<=N;i++)
      {
         sum=sum+term;
         term=term*4;
      }
      System.out.println("Sum="+sum);
   }
}
```

(vii)

```
import java.util.*;
public class ex42_vii_method1
{
   public static void main(String args[ ])
   {
      int sum,term1,term2,N;
      Scanner s=new Scanner(System.in);

      System.out.println("feed in N: ");
      N=s.nextInt();

      sum=0;
      term1=1;
      term2=3;
      for(int i=1;i<=N;i++)
      {
         sum=sum+term1*term2;
         term1=term1+1;
         term2=term2+3;
      }
      System.out.println("Sum="+sum);
   }
}
```

(viii)

```
import java.util.*;
public class ex42_viii_method1
{
   public static void main(String args[ ])
   {
      int sum1,term1,term2,N,sum2;
      double sum;
      Scanner s=new Scanner(System.in);

      System.out.println("feed in N: ");
      N=s.nextInt();

      sum1=0;
      term1=1;
      for(int i=1;i<=N;i++)
      {
         sum1=sum1+term1;
         term1=term1+2;
      }

      sum2=0;
      term2=2;
      for(int i=1;i<=N;i++)
      {
         sum2=sum2+term2;
         term2=term2+2;
      }

      sum=(double)sum1/sum2;
      System.out.println("Sum="+sum);
   }
}
```

(ix)

```
import java.util.*;
public class ex42_ix_method1
{
   public static void main(String args[ ])
   {
      int sum1,term1,term2,term3,term4,N,M,sum2;
      double sum;
      Scanner s=new Scanner(System.in);

      System.out.println("feed in N and M: ");
      N=s.nextInt();
      M=s.nextInt();

      sum1=0;
      term1=1;
      term2=3;
      for(int i=1;i<=N;i++)
      {
         sum1=sum1+term1*term2;
         term1=term1+1;
         term2=term2+3;
      }

      sum2=0;
      term3=1;
      term4=4;
      for(int i=1;i<=M;i++)
      {
         sum2=sum2+term3*term4;
         term3=term3+2;
         term4=term4+4;
      }
      sum=(double)sum1/sum2;
      System.out.println("Ans="+sum);
   }
}
```

**PE43:** Read N number and check how many are +ve, -ve and zero.

**Method 1**

```
import java.util.Scanner;
public class ex43_method1
{
   public static void main(String args[ ])
   {
      int i,N,pcount=0,ncount=0,zcount=0;
      double x;

      Scanner s=new Scanner(System.in);
      System.out.println("Feed in N: ");
      N=s.nextInt();

       for(i=1;i<=N;i++)
       {
         System.out.println("Feed in the number ");
         x=s.nextDouble();
         if (x>0)
               pcount++;
         if(x<0)
               ncount++;
         if(x==0)
               zcount++;
      }

       System.out.println("Positives="+pcount);
       System.out.println("Negatives="+ncount);
       System.out.println("Zeroes="+zcount);
   }
}
```

**Method 2**

```
import java.util.Scanner;
class COUNT
{
   private double x[ ];
   private int N,pcount,ncount,zcount;

   public void getdata()
   {
        Scanner s=new Scanner(System.in);
        System.out.println("Feed in N: ");
        N=s.nextInt();
        x=new double[N];
        System.out.println("feed in the elements ");
        for(int i=0;i<N;i++)
                x[i]=s.nextDouble();
   }

   public void calculate()
   {
      int i;
      pcount=ncount=zcount=0;//dont need to do this
      //By default all class variables are set to zero
      for(i=0;i<N;i++)
      {
         if (x[i]>0)
              pcount++;
         if(x[i]<0)
              ncount++;
         if(x[i]==0)
              zcount++;
      }
   }

   public void putdata()
   {
        System.out.println("Positives="+pcount);
        System.out.println("Negatives="+ncount);
        System.out.println("Zeroes="+zcount);
   }
}
```

```
public class ex43_method2
{
   public static void main(String args[ ])
   {
        COUNT X=new COUNT();
        X.getdata();
        X.calculate();
        X.putdata();
   }
}
```

**PE44:** Read N numbers are check how many are even and odd.

**Method 1**

```
import java.util.Scanner;
public class ex44_method1
{
   public static void main(String args[ ])
   {
      int i,N,ecount=0,ocount=0;
      double x;

      Scanner s=new Scanner(System.in);
      System.out.println("Feed in N: ");
      N=s.nextInt();

       for(i=1;i<=N;i++)
       {
         System.out.println("Feed in the number ");
         x=s.nextDouble();

         if (x%2==0)
             ecount++;
         else
            ocount++;
       }

      System.out.println("Even's="+ecount);
      System.out.println("Odd's="+ocount);
   }
}
```

**Method 2**

```
import java.util.Scanner;
class COUNT
{
   private double x[ ];
   private int N,ecount,ocount;

   public void getdata()
   {
        Scanner s=new Scanner(System.in);
        System.out.println("Feed in N: ");
        N=s.nextInt();
        x=new double[N];
        System.out.println("feed in the elements ");
        for(int i=0;i<N;i++)
            x[i]=s.nextDouble();
   }

   public void calculate()
   {
      int i;
      ecount=ocount=0;//dont need to do this
      //By default all class variables are set to zero
      for(i=0;i<N;i++)
      {
         if (x[i]%2==0)
                ecount++;
         else
                ocount++;
      }
   }

   public void putdata()
   {
      System.out.println("Even's="+ecount);
      System.out.println("Odd's="+ocount);
   }
}
```

```
public class ex44_method2
{
   public static void main(String args[ ])
   {
        COUNT X=new COUNT();
        X.getdata();
        X.calculate();
        X.putdata();
   }
}
```

**PE45:** Read N numbers & find maximum.

**Method 1**

```
import java.util.*;
public class ex45_method1
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      int N,i;
      double x,max=0;
      //giving max a dummy value since it is initialized
      //in the loop & used after too.

      System.out.println("Feed in the value of N: ");
      N=s.nextInt();


       for(i=1;i<=N;i++)
       {
         System.out.println("Feed in a number: ");
         x=s.nextDouble();

         if (i==1) //Its the first number
             max=x;
                 else if(x>max) //found a new max
                         max=x;
       }
              System.out.println("Maximum="+ max);
   }
}
```

**Method 2**

```
import java.util.*;
class MAX
{
   private int N;
   private double x[ ],max;

   public void getdata()
   {
        Scanner s=new Scanner(System.in);

        System.out.println("Feed in the value of N: ");
        N=s.nextInt();
        x=new double[N];

        System.out.println("Feed in the data ");
        for(int i=0;i<N;i++)
                x[i]=s.nextDouble();
   }

   public void calculate()
   {
      int i;
      for(i=0;i<N;i++)
      {
         if (i==0) //Its the first number
                        max=x[0];
         else if(x[i]>max) //found a new max
                        max=x[i];
      }
   }

    public void putdata()
   {
      System.out.println("Maximum="+ max);
   }
}
```

```
public class ex45_method2
{
   public static void main(String args[ ])
   {
        MAX X=new MAX();
        X.getdata();
        X.calculate();
        X.putdata();
   }
}
```

**PE46:**Read N numbers and find minimum

**Method 1**

```
import java.util.*;
public class ex46_method1
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      int N,i;
      double x,min=0;
      //giving min a dummy value since it is initialized
      //in the loop & used after too.

      System.out.println("Feed in the value of N: ");
      N=s.nextInt();

       for(i=1;i<=N;i++)
       {
              System.out.println("Feed in a number: ");
              x=s.nextDouble();

              if (i==1) //Its the first number
                     min=x;
               else
              {
                     if(x<min) //found a new min
                            min=x;
               }
       }
      System.out.println("Minimum="+ min);
   }
}
```

**Method 2**

```
import java.util.*;
class MAX
{
   private int N;
   private double x[ ],min;

   public void getdata()
   {
      Scanner s=new Scanner(System.in);
      System.out.println("Feed in the value of N: ");
      N=s.nextInt();
      x=new double[N];

       System.out.println("Feed in the data ");
       for(int i=0;i<N;i++)
              x[i]=s.nextDouble();
   }

   public void calculate()
   {
      int i;

      for(i=0;i<N;i++)
      {
              if (i==0) //Its the first number
                     min=x[0];
              else if(x[i]<min) //found a new min
                             min=x[i];
      }
   }

   public void putdata()
   {
      System.out.println("Minimum="+ min);
   }
}
```

```
public class ex46_method2
{
   public static void main(String args[ ])
   {
      MAX X=new MAX();
      X.getdata();
      X.calculate();
      X.putdata();
   }
}
```

**PE47:** Read N numbers and find the maximum, minimum & their positions.

**Method 1**

```
import java.util.*;
public class ex47_method1
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      int N,i,maxpos=0,minpos=0;
      double x,max=0,min=0;
     //giving min a dummy value since it is initialized in the loop & used after too.
      System.out.println("Feed in the value of N: ");
      N=s.nextInt();

      for(i=1;i<=N;i++)
      {
         System.out.println("Feed in a number: ");
         x=s.nextDouble();

         if (i==1) //Its the first number
         {
            max=x;
            min=x;
            maxpos=1;
            minpos=1;
         }
        else
         {
            if(x<min) //found a new min
            {
               min=x;
               minpos=i;
            }
            if(x>max) //found a new maximum
            {    max=x;
               maxpos=i;
            }
         }
      }//end of for
      System.out.println("Minimum="+ min + " and its at position="+minpos);
      System.out.println("Maximum="+ max+ " and its at position="+maxpos);
   }
}
```

**Method 2**

```
import java.util.*;
class MAX
{
   private int N,maxpos,minpos;
   private double x[ ],max,min;

   public void getdata()
   {
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in the value of N: ");
      N=s.nextInt();
      x=new double[N];

      System.out.println("Feed in the data ");
      for(int i=0;i<N;i++)
               x[i]=s.nextDouble();
    }

   public void calculate()
   {
      int i;
      max=0;//DON'T NEED TO DO THIS, CLASS VARIABLES ARE AUTOSET TO 0
      min=0;//DON'T NEED TO DO THIS, CLASS VAARIABLES ARE AUTOSET TO 0

      for(i=0;i<N;i++)
      {
         if (i==0) //Its the first number
         {
            max=x[0];
            maxpos=0;
            min=x[0];
            minpos=0;
         }
```

```
        else
        {
           if(x[i]<min) //found a new min
           {  min=x[i];
              minpos=i;
           }
           if(x[i]>max) //found a new max
           {  max=x[i];
              maxpos=i;
           }
        }
      }
   }

   public void putdata()
   {
       System.out.println("Minimum="+ min + " at position "+minpos);
       System.out.println("Maximum="+ max + " at position "+maxpos);
   }
}

public class ex47_method2
{
   public static void main(String args[ ])
   {
       MAX X=new MAX();
       X.getdata();
       X.calculate();
       X.putdata();
   }
}
```

**PE48:** Read N numbers & find the highest & second highest.

**Method 1**

```
import java.util.Scanner;
public class ex48_method1
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      int N,i;
      double x,max1,max2;
      System.out.println("Feed in N: ");
      N=s.nextInt();

       max1=max2=0;//REQUIRED SINCE IT'S NOT 2 CLASS METHOD
       System.out.println("feed in the numbers");
       for(i=1;i<=N;i++)
       {
              x=s.nextDouble();
              if (i==1)
                     max1=x;
              if(i==2)
              {
                  if(x>max1)
                  {
                     max2=max1;
                     max1=x;
                  }
                  else
                    max2=x;
                }

         if (i>=3)
        {
           if(x>max1)
           {   max2=max1;
              max1=x;
           }
           else if(x>max2)
                     max2=x;
         }
      }//end of for
       System.out.println("Maximum="+max1);
       System.out.println("2nd Maximum="+max2);
```

```
    }
}
```

**Method 2**

```
import java.util.Scanner;
class MAXMAX
{
   private double x[ ],max1, max2;
   private int N;

   public void getdata()
   {
        Scanner s=new Scanner(System.in);
        System.out.println("Feed in N: ");
        N=s.nextInt();
        x=new double[N];

        System.out.println("Feed in the numbers ");
        for(int i=0;i<N;i++)
                x[i]=s.nextDouble();
   }

   public void calculate()
   {
      int i;
      for(i=0;i<N;i++)
      {
         if (i==0)
            max1=x[i];
         if(i==1)
         {

            if(x[i]>max1)
            {
               max2=max1;
               max1=x[i];
            }
            else
               max2=x[i];
         }
```

```
        if (i>=2)
        {
           if(x[i]>max1)
           {
              max2=max1;
              max1=x[i];
           }
           else if(x[i]>max2)
              max2=x[i];
        }
      }//end of for
   }

   public void putdata()
   {
      System.out.println("Maximum="+max1);
      System.out.println("2nd Maximum="+max2);
   }
}


public class ex48_method2
{
   public static void main(String args[ ])
   {
      MAXMAX X=new MAXMAX();
      X.getdata();
      X.calculate();
      X.putdata();
   }
}
```

**PE49:** Check if a number is prime or not.

**Method 1**

```
import java.util.*;
public class ex49_method1
{
   public static void main(String args[ ])
   {
      int N,i;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in N ");
      N=s.nextInt();

       for(i=2;i<=N-1;i++)
       {
        if (N%i==0)
           break;
       }

       if(i<=N-1)
                    System.out.println("Number is NOT Prime");
       else
                    System.out.println("No is PRIME");
    }
}
```

**Method 2**

```
import java.util.*;
public class ex49_method2
{
   public static void main(String args[ ])
   {
      int N,i;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in N ");
      N=s.nextInt();

       for(i=2;i<=N-1;i++)
       {
         if (N%i==0)
         {
            System.out.println("Number is NOT Prime");
             return;
         }
       }
      System.out.println("No is PRIME");
   }
}
```

**Method 3**

```
import java.util.*;
public class ex49_method3
{
   public static void main(String args[ ])
   {
      int N,i;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in N ");
      N=s.nextInt();

       for(i=2;i<=N-1;i++)
       {
         if (N%i==0)
         {
            System.out.println("Number is NOT Prime");
            System.exit(0);
         }
       }
      System.out.println("No is PRIME");
   }
}
```

**PE50:** Print all prime numbers from 1 to N

```
import java.util.*;
public class ex50_method1
{
   public static void main(String args[ ])
   {
      int N,n,i;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in N ");
      N=s.nextInt();

       for(n=2;n<=N;n++) //choosing a number
       {
        for(i=2;i<=n-1;i++)
        {
                    if (n%i==0)
                    break;
        }

        if(i<=n-1)
        { }
       else
             System.out.println(n);
       }
   }
}
```

**PE51:** Print all perfect squares from 1 to N

```
import java.util.*;
public class ex51_method1
{
   public static void main(String args[ ])
   {
      int i, N;
      Scanner s=new Scanner(System.in);

      System.out.println("feed in N: ");
      N=s.nextInt();

       for(i=1;i<N;i++)
       {
         if (i*i<=N)
         {
            System.out.println(i*i);
         }
      }
   }
}
```

**PE52:** Print a rectangle of * with 60 *'s on each row & totally n rows, where n is a user input.

```
import java.util.*;
public class ex52
{
   public static void main(String args[ ])
   {
      int row,star,N;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in number of rows");
      N=s.nextInt();

       for(row=1;row<=N;row++)
       {
         for(star=1;star<=60;star++)
         {
            System.out.print("*");
         }
         System.out.println();
      }
   }
}
```

**PE53:** Write a program to print the following:

(a)
```
*
**
***
****
*****
```
(b)
```
*****
****
***
**
*
```
(c)
```
*****
 ****
  ***
   **
    *
```

(a)

```
import java.util.*;
public class ex53_a_method1
{
   public static void main(String args[ ])
   {
      int row,star,N;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in number of rows");
      N=s.nextInt();

       for(row=1;row<=N;row++)
       {
         for(star=1;star<=row;star++)
         {
            System.out.print("*");
         }
         System.out.println();
      }
   }
}
```

(b)

```
import java.util.*;
public class ex53_b_method1
{
   public static void main(String args[ ])
   {
      int row,star,N;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in number of rows");
      N=s.nextInt();

       for(row=N;row>=1;row--)
       {
         for(star=1;star<=row;star++)
         {
            System.out.print("*");
         }
         System.out.println();
       }
   }
}
```

(c)

```
import java.util.*;
public class ex53_c_method1
{
   public static void main(String args[ ])
   {
      int row,star,N,sp,i;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in number of rows");
      N=s.nextInt();

      for(sp=0,row=N;row>=1;row--,sp++)
      {
         for(i=1;i<=sp;i++)
         {
            System.out.print(" ");
         }
         for(star=1;star<=row;star++)
         {
            System.out.print("*");
         }
         System.out.println();
      }
   }
}
```

PE53 cont….: Write a program to print the following:

(d)
```
    *
   **
  ***
 ****
*****
```

(e)
```
    *
   ***
  *****
 *******
*********
```

(f)
```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

(d)

```
import java.util.*;
public class ex53_d_method1
{
   public static void main(String args[ ])
   {
      int row,star,N,sp,i;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in number of rows");
      N=s.nextInt();

       for(sp=N-1,row=1;row<=N;row++,sp--)
       {
         for(i=1;i<=sp;i++)
       System.out.print(" ");

         for(star=1;star<=row;star++)
            System.out.print("*");

         System.out.println();
      }
   }
}
```

(e)

```
import java.util.*;
public class ex53_e_method1
{
   public static void main(String args[ ])
   {
      int row,star,N,sp,i;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in number of rows");
      N=s.nextInt();

       for(sp=N-1,row=1;row<=N;row++,sp--)
       {
         for(i=1;i<=sp;i++)
               System.out.print(" ");

         for(star=1;star<=2*row-1;star++)
            System.out.print("*");

         System.out.println();
      }
   }
}
```

(f)

```
import java.util.*;
public class ex53_f_method1
{
   public static void main(String args[ ])
   {
      int row,star,N,sp,i;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in number of rows");
      N=s.nextInt();

       for(sp=40,row=1;row<=N/2;row++,sp--)
       {
         for(i=1;i<=sp;i++)
            System.out.print(" ");

         for(star=1;star<=2*row-1;star++)
            System.out.print("*");

         System.out.println();
       }

       if (N%2==0)  sp=sp+1;

       for(row=N-N/2;row>=1;row--,sp++)
       {
         for(i=1;i<=sp;i++)
               System.out.print(" ");

         for(star=1;star<=2*row-1;star++)
               System.out.print("*");
         System.out.println();
      }
   }
}
```

DESIGNS……

DESIGNS……

**PE54:** Write a program to display the PASCAL's triangle

```
                1           1
            1           2           1
        1           3           3           1
1           4           6           4           1
……..
```

The number of rows will be user input.

```
import java.util.*;
public class ex54_method1
{
   public static int fact(int n)
   {
      int p=1,i;
      for(i=1;i<=n;i++)
         p=p*i;
      return p;
   }

   public static void main(String args[ ])
   {
      int N,n,r,sp,i;
      Scanner s=new Scanner(System.in);

     System.out.println("Feed in the number of rows:");
     N=s.nextInt();
      for(sp=40,n=1;n<=N;n++,sp=sp-2)
      {
         for(i=1;i<=sp;i++)
                  System.out.print(" ");

         for(r=0;r<=n;r++)
                  System.out.print(fact(n)/(fact(r)*fact(n-r))+"   ");

         System.out.flush(); //Forcing output onto screen
         System.out.println();
      }
   }
}
```

**PE55:** Finding the sum of digits of a number

```
import java.util.*;
public class ex55_method1
{
   public static void main(String args[ ])
   {
      int N,sum=0,digit;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in the number:");
      N=s.nextInt();

       while(N!=0)
       {
         digit=N%10;
         N=N/10;
         sum=sum+digit;
       }

       System.out.println("Sum of digits="+sum);
   }
}
```

**PE56:** Finding product of digits of a number.

```
import java.util.*;
public class ex56_method1
{
   public static void main(String args[ ])
   {
      int N,p=1,digit;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in the number:");
      N=s.nextInt();

       while(N!=0)
       {
         digit=N%10;
         N=N/10;
         p=p*digit;
       }
      System.out.println("Product of digits="+p);
   }
}
```

**PE57:** Finding the sum of squares of the digits of a number.

```
import java.util.*;
public class ex57_method1
{
   public static void main(String args[ ])
   {
      int N,sum=0,digit;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in the number:");
      N=s.nextInt();

       while(N!=0)
       {
         digit=N%10;
         N=N/10;
         sum=sum+digit*digit;
       }

      System.out.println("Sum of squares digits="+sum);
   }
}
```

**PE58:** Finding the sum of cubes of the digits of a number.

```
import java.util.*;
public class ex58_method1
{
   public static void main(String args[ ])
   {
      int N,sum=0,digit;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in the number:");
      N=s.nextInt();

       while(N!=0)
       {
         digit=N%10;
         N=N/10;
         sum=sum+digit*digit*digit;
       }

      System.out.println("Sum of cubes digits="+sum);
   }
}
```

**PE59:** Check whether a number is a Armstrong number or not.

```
import java.util.*;
public class ex59_method1
{
   public static void main(String args[ ])
   {
      int N,sum=0,digit,x,c,y;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in the number:");
      N=s.nextInt();

       y=N;
       c=0;
       while(y!=0)
       {
              y=y/10;
              c=c+1;
       }

       x=N;
       while(N!=0)
       {
         digit=N%10;
         N=N/10;
         sum=sum + Math.pow(digit,c);
       }

       if (x==sum)
              System.out.println("Number is an Armstrong Number");
       else
              System.out.println("NOT a Armstrong Number");
   }
}
```

**PE60:** Print all Armstrong numbers from 1 to N.

```
import java.util.*;
public class ex60_method1
{
   public static void main(String args[ ])
   {
      int N,sum=0,digit,x,n,y,c;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in N: ");
      N=s.nextInt();

       for(x=1;x<=N;x++)
       {
              y=x;
              c=0;
              while(y!=0)
              {
                     y=y/10;
                     c=c+1;
              }

              n=x;
              sum=0;
               while(n!=0)
        {
                     digit=n%10;
                     n=n/10;
                     sum=sum+digit*digit*digit;
                }

              if (x==sum)
                     System.out.println(x);
         }
   }
}
```

**PE61:** Print reverse of a number.

```
import java.util.*;
public class ex61_method1
{
   public static void main(String args[ ])
   {
      int N,r=0,digit;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in the number:");
      N=s.nextInt();

       while(N!=0)
       {
         digit=N%10;
         N=N/10;
         r=r*10+digit;
       }

      System.out.println("reverse="+r);
   }
}
```

**PE62:** Write a menu driven program to perform the basic operations of a calculator.

**PE63:** that determines the GCD of two positive integers.

```
import java.util.Scanner;
public class ex63_method1
{
   public static void main(String args[ ])
   {
      int m,n,i;
      Scanner s=new Scanner(System.in);
      System.out.println("Feed in the 2 numbers");
      m=s.nextInt();
      n=s.nextInt();

       for(i=m;i>=1;i--)
       {
         if (m%i==0&&n%i==0)
            break;
       }
      System.out.println("GCD="+i);
   }
}
```

**PE64:** Write a program that determines the GCD by Euclidean algorithm. (The algorithm begins by dividing the 1st number by the 2nd and retaining the remainder. At each successive stage, the previous divisor is divided by the previous remainder. The algorithm stops when the remainder becomes zero. The GCD is the last non-zero remainder (or the last divisor)).

```
import java.util.Scanner;
public class ex64_method1
{
   public static void main(String args[ ])
   {
      int m,n,rem;
      Scanner s=new Scanner(System.in);
      System.out.println("Feed in the 2 numbers");
      m=s.nextInt();
      n=s.nextInt();

       while(true)
       {
         rem=m%n;

         if (rem==0)
            break;

         m=n;
         n=rem;
       }

      System.out.println("GCD="+n);
   }
}
```

**PE65:** Write a program that reads a positive integer & classifies it as being deficient, perfect or abundant. A number is said to be deficient, perfect or abundant if the sum of its proper divisors is less than the number, equal to the number or more than the numbers. Example: 4 is deficient since 1+2 <4; 6 is perfect since 1+2+3=6; 12 is abundant since 1+2+3+4+6 > 12.

```
import java.util.*;
public class ex65_method1
{
   public static void main(String args[ ])
   {
      int N,i,sum=0;
      Scanner s=new Scanner(System.in);

      System.out.println("feed in the number");
      N=s.nextInt();

       for(i=1;i<N;i++)
       {
         if (N%i==0)
              sum=sum+i;
        }

       if (sum==N)
              System.out.println("Number is perfect");
       if (sum<N)
              System.out.println("Number is deficient");
       if (sum>N)
              System.out.println("Number is abundant");
       }
}
```

**PE66:** Three integers a,b and c are such that they form a Pythagorean triplet i.e. $a^2+b^2=c^2$. Write a program that generates all Pythagorean triplets a,b,c where a,b,c ≤ 25.

```
import java.util.*;
public class ex66_method1
{
   public static void main(String args[ ])
   {
      int a,b,c;

      for(a=1;a<=25;a++)
            for(b=1;b<=25;b++)
                  for(c=1;c<=25;c++)
                        if (a*a+b*b==c*c)
                              System.out.println(a+","+b+","+c);
   }
}
```

**PE67:** Modify the above program to generate triplets a,b,c such that a < b <c and a,b ≤ 25.

```
import java.util.*;
public class ex67_method1
{
   public static void main(String args[ ])
   {
         int a,b,c;

      for(a=1;a<=25;a++)
            for(b=1;b<=25;b++)
                  for(c=1;;c++)
                        if (a*a+b*b<c*c)   break;

       if (a<b&& b<c && a*a+b*b==c*c)
            System.out.println(a+","+b+","+c);
   }
}
```

**PE68:** Write a program to calculate sin(x) and cos(x) using the expansion series formula. The user will enter up to which term the calculation needs to proceed

**PE69:** Write a program to calculate sin(x) and cos(x) using the expansion series formula. The user will enter the accuracy up to which the result should be displayed in.

# Practice Questions

1. Calculate a raised to b without pow, assume a & b are integers.
2. Convert any given number to ROMAN

| Decimal | ROMAN |
|---|---|
| 1 | I |
| 5 | V |
| 10 | X |
| 50 | L |
| 100 | C |
| 500 | D |
| 1000 | M |

Example Roman equivalent of 1988 is MDCCCCLXXXVIII

3. A positive integer is entered through the keyboard. Write a program to obtain the prime factors of this number. E.g. prime factors of 24 are 2, 2, 2 and 3, whereas prime factors of 35 are 5 and 7.
4. Write a program to obtain the first N terms of the Fibonacci sequence 1,1,2,3,5,8,13,21,34,…….
5. Write a program to print the binary equivalent of a number.
6. Find the sum of the series
   $\sin(x) = x - (x^3/3!) + (x^5/5!) - (x^7/7!) + \ldots\ldots$ N terms
7. Find the sum of the following series:
   $\frac{1}{1!} + \frac{2}{2!} + \frac{3}{3!} + \ldots\ldots$ N terms
8. Write a program to generate all combinations of 1, 2 and 3
9. According to a study, the approximate level of intelligence of a person can be calculated using the following formula:
   $i = 2 + (y + 0.5x)$
   Write a program that produces a table os values of i, y, x where y varies from 1 to 6, and, for each value of y, x varies from 5.5 to 12.5 in steps of 0.5
10. Suppose you place a given sum of money, A, into a saving account at the beginning of each year for n years. If the account earns interest at the rate of i% anually, then the amount of money that has accumulated after n years, F is given by
    $F=A[(1+i/100) + (1+i/100)^2 + (1+i/100)^3 + \ldots\ldots + (1+i/100)^n]$
    Write an interactive program to determine the following:
    i. How much money gets accumulated after n years
    ii. How much money needs to be deposited at the beginning of the five year to get an amount $F_{amt}$.
11. Write a program which finds four digit perfect squares, where the number represented by the first two digits and last two digits are also perfect squares. Example: $1681 = 41^2$, $16=4^2$, $81=9^2$
12. Write a program that reads a number with a decimal point & it prints out the number of digits to the left & right of the decimal point.

13. Write a program to find the sum of prime numbers between the limits specified by the user.
14. Read numbers from a user till he enters a negative value. Find the number of values entered, the highest & the least
15. Mohan has invested Rs.1000/- at 10% simple interest, wheras Vidhya has invested Rs.750 at 7.5% interested compounded annually. Write a program to determine the number of years it will take for the value of Vidhya's investment to exceed that of Mohan's.
16. Write a program to convert a binary number to decimal number system.
17. Print the Fibonacci series of N terms
18. Print the Fibonacci series upto N
19. Read a number from the user & check if it is a Fibonacci number or not.
20. Write a program to generate every third positive integer from 2 to 100 & find the sum of the generated integers which are divisible by 5.
21. An object starting at rest and travels a diatance of 'S' given by $S=at^2/2$. 'a' changes by 2 $m/s^2$ in the range of $1 \le t \le 10$ and decrements by 1$m/s^2$ for next 5sec. Write a programs that prints a table of time, acceleration and distance, where time is to be taken in intervals of 1s.
22. Write a program that checks if two numbers are same upto 2 decimal place.
23. Write a program that prints out all the factors of a number e.g. for 150 the output should show
    2
    3
    5
    5
24. Write a program to compute the distance S fallen by an object in free fall. The formula is
    $S=So+Vo \cdot t+1/2.at^2$
    Make a table of S for t=1,5,10,15,……100.
25. Write a program that, give a input number, prints its square , cube, and fourth power without doing any unnecessary calculations.
26. A famous conjecture holds that all positive integers converge to 1 (one) when treated in the following fashion.
    i. If the number is odd, it is multiplied by 3 and 1 is added.
    ii. If the number is even, it is divided by 2
    iii. Continuously apply the above operations to the intermediate results until the number converges to 1.

    Write a program to read an integer from the keyboard & implement the above mentioned algorithm and display all the immediate values until the number converges to 1. Also count the number of iterations required for convergence.

# Chapter 4

# 1 D Numeric Arrays

**THE DIFFERENCE**

The main difference between arrays in C++ and arrays in Java is that arrays are treated like objects hence memory creation should be dynamic.

e.g. int x[ ]=new int[10];

NOTE: dynamic array creation allows users to create arrays of user defined size hence eliminating the use of pointers

e.g. int x[ ]=new int[n];

**When to use Arrays**

If a large number of memory locations (variables) are needed & all are of the same type then arrays can be used.

e.g. Say we want to read 100 integers. Now we need 100 memory locations

int a,b,c,d,e,…….;

a=s.nextInt();

b=s.nextInt(); ……. And so on

Above method is NOT PRACTICAL

**Reading into an Array**

```
int x[ ]=new int[100];
for(i=0;i<100;i=i+1)
{
        x[i]=s.nextInt();
}
```

NOTE: array size is 100. All are integers. First variable is called x[0] & the last is called x[99].

**Printing an Array**

We have to print x[0], then x[1], then x[2]….. Till x[99]

AGAIN LOOPs…..

```
for(i=0;i<100;i=i+1)
{
        System.out.print(x[i]+"\t");
}
```

**PE70:** Write a program to read 100 numbers and find the sum and average.

**PE71:** Write a program to read N number and print them. Also display the sum & average

**Method 1**

```
import java.util.Scanner;
public class ex71_method1
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      double x[ ],sum=0,avg;
      int N;

      System.out.println("Feed in the number of elements");
      N=s.nextInt();

       x=new double[N];

      System.out.println("Feed in the N elements");
      for(int i=0;i<N;i++)
            x[i]=s.nextDouble();

       for(int i=0;i<N;i++)
            sum=sum+x[i];

       avg=sum/N;

       System.out.println("Sum="+sum);
       System.out.println("Average="+avg);
   }
}
```

**Method 2**

```
import java.util.*;
class SA
{
   private int N;
   private double x[ ],sum,avg;
   public void getdata()
   {
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in the number of elements");
      N=s.nextInt();

       x=new double[N];

      System.out.println("Feed in the N elements");
      for(int i=0;i<N;i++)
               x[i]=s.nextDouble();
    }
   public void calculate()
   {
      sum=0; //NOT REQUIRED
      for(int i=0;i<N;i++)
               sum=sum+x[i];

      avg=sum/N;
   }
   public void putdata()
   {
        System.out.println("Sum="+sum);
        System.out.println("Average="+avg);
   }
}

public class ex71_method2
{  public static void main(String args[ ])
   {  SA X=new SA();
      X.getdata();
      X.calculate();
      X.putdata();
   }
}
```

**PE72:** Write a program to read N number and print how many are above average and how many are below average and print the two lists.

```
import java.util.*;
public class ex72_method1
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      double x[ ],sum=0, aa=0,ba=0,avg;
      int N;

      System.out.println("Feed in the number of elements");
      N=s.nextInt();

       x=new double[N];

      System.out.println("Feed in the N elements");
      for(int i=0;i<N;i++)
              x[i]=s.nextDouble();

      for(int i=0;i<N;i++)
              sum=sum+x[i];

      avg=sum/N;

       for(int i=0;i<N;i++)
       {
         if (x[i]>avg)
             aa++;
         if (x[i]<avg)
                ba++;
       }


       System.out.println("Sum="+sum);
       System.out.println("Average="+avg);
       System.out.println("Above Average="+aa+ " List is ");
```

```
        for(int i=0;i<N;i++)
        {
          if (x[i]>avg)
             System.out.println(x[i]);
        }

        System.out.println("Below Average="+ba+ " List is ");
        for(int i=0;i<N;i++)
        {
          if (x[i]<avg)
                    System.out.println(x[i]);
        }
   }
}
```

**Method 2**

```
import java.util.*;
class AABA
{
   private int N;
   private double x[ ],sum,avg,aa,ba;

   public void getdata()
   {
      Scanner s=new Scanner(System.in);
      System.out.println("Feed in the number of elements");
      N=s.nextInt();

      x=new double[N];

       System.out.println("Feed in the N elements");
       for(int i=0;i<N;i++)
       {
         x[i]=s.nextDouble();
       }
    }
```

```
   public void calculate()
   {   sum=0; //NOT REQUIRED
       aa=ba=0;//NOT REQUIRED

       for(int i=0;i<N;i++) sum=sum+x[i];

       avg=sum/N;

       for(int i=0;i<N;i++)
       {
          if (x[i]>avg)   aa++;
          if(x[i]<avg)    ba++;
       }
   }

   public void putdata()
   {
                System.out.println("Sum="+sum);
                System.out.println("Average="+avg);

                System.out.println("Above Average="+aa + " List is" );
                for(int i=0;i<N;i++)
                        if (x[i]>avg) System.out.println(x[i]);

                System.out.println("Below Average="+ba+" List is ");
                for(int i=0;i<N;i++)
                        if (x[i]<avg)   System.out.println(x[i]);
    }
}

public class ex72_method2
{
   public static void main(String args[ ])
   {
       AABA X=new AABA();
       X.getdata();
       X.calculate();
       X.putdata();
   }
}
```

**PE73:** Write a program to read N marks and prepare a frequency distribution table 0-40, 40-60, 60-80,80-100

**Method 1**

```
import java.util.Scanner;
public class ex73_method1
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      int N,a=0,b=0,c=0,d=0,x;

      System.out.println("How many data item do you have?");
      N=s.nextInt();

      for(int i=0;i<N;i++)
      {
         System.out.println("feed in a value ");
         x=s.nextInt();
         if (x<40)     a++;
                     else if(x<60) b++;
                             else if(x<80) c++;
                                     else if(x<=100)  d++;
       }

       System.out.println("0-40\t"+a);
       System.out.println("40-60\t"+b);
       System.out.println("60-80\t"+c);
       System.out.println("80-100\t"+d);
   }
}
```

**Method 2**

```
import java.util.*;
class FREQ
{
   private int N,x[ ],a,b,c,d;
   public void getdata()
   {
        Scanner s=new Scanner(System.in);
        System.out.println("How many data item do you have?");
        N=s.nextInt();
        x=new int[N];
        System.out.println("feed in the elemnts");
        for(int i=0;i<N;i++)
               x[i]=s.nextInt();
        }

   public void calculate()
   {  //a=b=c=d=0; //NOT REQUIRED IN 2 CLASS METHOD
      for(int i=0;i<N;i++)
      {
         if (x[i]<40) a++;
                  else if(x[i]<60)  b++;
                        else if(x[i]<80)  c++;
                              else if(x[i]<=100)   d++;
       }
   }

   public void putdata()
   {  System.out.println("0-40\t"+a);
      System.out.println("40-60\t"+b);
      System.out.println("60-80\t"+c);
      System.out.println("80-100\t"+d);
   }
}
public class ex73_method2
{  public static void main(String args[ ])
   {  FREQ X=new FREQ();
      X.getdata();
      X.calculate();
      X.putdata();
   }
}
```

**PE74:** Write a program to read N number and cycle the elements of the array i.e. 1st move into 2nd ,2nd moves into 3rd…… last moves into 1st slot.

**Method 1**

```
import java.util.Scanner;
public class ex74_method1
{
   public static void main(String args[ ])
   {
      double x[ ];
      int N;

      Scanner s=new Scanner(System.in);
      System.out.println("Enter N");
      N=s.nextInt();

      System.out.println("Feed in Array");
      x=new double[N];

      for(int i=0;i<N;i++)
               x[i]=s.nextDouble();

      double t=x[N-1];
      for( int i=N-2;i>=0;i--)
                     x[i+1]=x[i];

         x[0]=t;

      for(int i=0;i<N;i++)
                     System.out.println(x[i]);
   }
}
```

**Method 2**

```
import java.util.Scanner;
class SHIFT
{
   private double x[ ];
   private int N;

   public void getdata()
   {  Scanner s=new Scanner(System.in);
      System.out.println("Enter N");
      N=s.nextInt();
      System.out.println("Feed in Array");
      x=new double[N];
      for(int i=0;i<N;i++)
                        x[i]=s.nextDouble();
    }

   public void calculate()
   {
      double t=x[N-1];
      for( int i=N-2;i>=0;i--)
      {
         x[i+1]=x[i];
      }
      x[0]=t;
   }

   public void putdata()
   {
      for(int i=0;i<N;i++)
                        System.out.println(x[i]);
   }
}
public class ex74_method2
{  public static void main(String args[ ])
   {  SHIFT X=new SHIFT();
      X.getdata();
      X.calculate();
      X.putdata();
   }
}
```

**PE75:** Write a program to read N numbers and swap the adjacent elements i.e. swap 1st & 2nd ,3rd and 4th,……

**Method 1**

```
import java.util.*;
public class ex75_method1
{
   public static void main(String args[ ])
   {
      double x[ ],t;
      int N;

      Scanner s=new Scanner(System.in);
      System.out.println("Enter N");
      N=s.nextInt();
      System.out.println("Feed in Array");
      x=new double[N];

          for(int i=0;i<N;i++)
                    x[i]=s.nextDouble();

      for(int i=0;i<=N-2;i=i+2)
      {
         t=x[i];
         x[i]=x[i+1];
         x[i+1]=t;
      }

      for(int i=0;i<N;i++)
                    System.out.println(x[i]);
       }
}
```

**Method 2**

```
import java.util.Scanner;
class SHIFT
{   double x[ ];
    int N;

    public void getdata()
    {   Scanner s=new Scanner(System.in);
        System.out.println("Enter N");
        N=s.nextInt();
        System.out.println("Feed in Array");
        x=new double[N];
        for(int i=0;i<N;i++)
                x[i]=s.nextDouble();
         }

    public void calculate()
    {
        double t;
        for( int i=0;i<=N-2;i=i+2)
        {
            t=x[i];
            x[i]=x[i+1];
            x[i+1]=t;
        }
    }

    public void putdata()
    {
         for(int i=0;i<N;i++)
                System.out.println(x[i]);
    }
}

public class ex75_method2
{   public static void main(String args[ ])
    {   SHIFT X=new SHIFT();
        X.getdata();
        X.calculate();
        X.putdata();
    }
}
```

**PE76:** Write a program to read N number and find an element in that list using linear or sequential search

**Method 1**

```
import java.util.*;
public class ex76_method1
{
   public static void main(String args[ ])
   {
      double x[ ],target;
      int i,N;
      Scanner s=new Scanner(System.in);

      System.out.println("Enter N");
      N=s.nextInt();

         System.out.println("Feed in Array");
      x=new double[N];

      for(i=0;i<N;i++)
      {
         x[i]=s.nextDouble();
      }

      System.out.println("Feed in element to search");
      target=s.nextDouble();

         for(i=0;i<N;i++)
      {
        if (x[i]==target)
          break;
      }

      if (i<N)
         System.out.println("Found at position="+i);
      else
         System.out.println("Not found");
   }
}
```

**Method 2**

```
import java.util.Scanner;
class LINEAR
{
   double x[ ],target;
   int N;
   public void getdata()
   {   Scanner s=new Scanner(System.in);
      System.out.println("Enter N");
      N=s.nextInt();
      System.out.println("Feed in Array");
      x=new double[N];
       for(int i=0;i<N;i++)
       {
         x[i]=s.nextDouble();
       }

      System.out.println("Feed in the element to search: ");
      target=s.nextDouble();
   }
   public void putdata()
   {
      int i;
      for(i=0;i<N;i++)
      {  if (x[i]==target)
            break;
      }
      if(i<N)
         System.out.println("Found at location "+i);
      else
         System.out.println("NOT FOUND");
   }
}
public class ex76_method2
{  public static void main(String args[ ])
   {
      LINEAR X=new LINEAR();
      X.getdata();
      X.putdata();
   }
}
```

**PE77:** Write a program to read N numbers and find an element using binary search

```
import java.util.*;
public class ex77_method1
{  public static void main(String args[ ])
   {
        double x[ ],target;
        int i,N,low,high,mid;
        Scanner s=new Scanner(System.in);
        System.out.println("Enter N");
        N=s.nextInt();
        System.out.println("Feed in Array");
        x=new double[N];
        for(i=0;i<N;i++)
           x[i]=s.nextDouble();

      System.out.println("Feed in element to search");
      target=s.nextDouble();

      low=0;
      high=N-1;

      mid=0; //NECESSARY IN ONE CLASS METHOD
       while (low<=high)
       {
         mid=(low+high)/2;
         if(target==x[mid])  break;
         if(target<x[mid])  high=mid-1;
         if(target>x[mid])   low=mid+1;
      }

        if (low<=high) System.out.println("Found at position="+mid);
              else  System.out.println("Not found");
   }
}
```

**PE78:** Write a program to read N number and arrange in ascending order using bubble sort

**Method 1**

```
import java.util.*;
public class ex78_method1
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      double x[ ],t;
      int N,pos,cpos;

      System.out.println("Feed in the number of elements");
      N=s.nextInt();

      x=new double[N];
          System.out.println("Feed in the N elements");
      for(int i=0;i<N;i++)
                    x[i]=s.nextDouble();

      for(pos=0;pos<=N-2;pos++)
      {
              for(cpos=pos+1;cpos<=N-1;cpos++)
              {
                        if (x[pos]>x[cpos])
                        {
                                t=x[pos];
                                x[pos]=x[cpos];
                                x[cpos]=t;
                        }
              }
      }

        System.out.println("Sorted List is");
        for(int i=0;i<N;i++)
        {
          System.out.println(x[i]);
        }
   }
}
```

**Method 2**

```
import java.util.*;
class BUBBLE
{  private double x[ ],t;
   private int N;
   public void getdata()
   {  Scanner s=new Scanner(System.in);
      System.out.println("Feed in the number of elements");
      N=s.nextInt();
          x=new double[N];
      System.out.println("Feed in the N elements");
      for(int i=0;i<N;i++)
         x[i]=s.nextDouble();
   }

    public void calculate()
   {  int pos,cpos;
      for(pos=0;pos<=N-2;pos++)
      {  for(cpos=pos+1;cpos<=N-1;cpos++)
         {  if (x[pos]>x[cpos])
            {  t=x[pos];
               x[pos]=x[cpos];
               x[cpos]=t;
            }
         }
      }
   }
   public void putdata()
   {
      System.out.println("Sorted List is");
      for(int i=0;i<N;i++)
         System.out.println(x[i]);
   }
}
public class ex78_method2
{   public static void main(String args[ ])
   {  BUBBLE X=new BUBBLE();
      X.getdata();
      X.calculate();
      X.putdata();
   }
}
```

**PE79:** Write a program to read N number and arrange in descending order using bubble sort

**Method 1**

```
import java.util.*;
public class ex79_method1
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      double x[ ],t;
      int N,pos,cpos;

      System.out.println("Feed in the number of elements");
      N=s.nextInt();
      x=new double[N];

      System.out.println("Feed in the N elements");
      for(int i=0;i<N;i++)
      {
         x[i]=s.nextDouble();
      }

          for(pos=0;pos<=N-2;pos++)
      {
                    for(cpos=pos+1;cpos<=N-1;cpos++)
            {
                          if (x[pos]<x[cpos])
            {
                                t=x[pos];
                    x[pos]=x[cpos];
                    x[cpos]=t;
            }
            }
      }
      System.out.println("Sorted List is");
      for(int i=0;i<N;i++)
            System.out.println(x[i]);
       }
}
```

**Method 2**

```
import java.util.*;
class BUBBLE
{   private double x[ ],t;
    private int N;
    public void getdata()
    {   Scanner s=new Scanner(System.in);
        System.out.println("Feed in the number of elements");
        N=s.nextInt();
        x=new double[N];
        System.out.println("Feed in the N elements");
        for(int i=0;i<N;i++)
            x[i]=s.nextDouble();
    }
    public void calculate()
    {   int pos,cpos;
        for(pos=0;pos<=N-2;pos++)
        {   for(cpos=pos+1;cpos<=N-1;cpos++)
            {   if (x[pos]<x[cpos])
                {
                    t=x[pos];
                    x[pos]=x[cpos];
                    x[cpos]=t;
                }
            }
        }
    }
    public void putdata()
    {   System.out.println("Sorted List is");
        for(int i=0;i<N;i++)
            System.out.println(x[i]);
    }
}
public class ex79_method2
{    public static void main(String args[ ])
    {   BUBBLE X=new BUBBLE();
        X.getdata();
        X.calculate();
        X.putdata();
    }
}
```

**PE80:** Write a program to read N number and arrange in ascending order using selection sort

**Method 1**

```
import java.util.*;
public class ex80_method1
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      double x[ ],large;
      int N,pos,i,j;

      System.out.println("Feed in the number of elements");
      N=s.nextInt();
      x=new double[N];

      System.out.println("Feed in the N elements");
      for(i=0;i<N;i++)
      {
         x[i]=s.nextDouble();
      }

          for(i=N-1;i>0;i--) //where to put large & upto where to compare
      {   large=x[0];
         pos=0;         //ASSUMPTION
         for(j=1;j<=i;j++)   //COMPARISON
         {
            if (x[j]>large)
            {   large=x[j];
               pos=j;
            }
         }
         x[pos]=x[i];
         x[i]=large;
      }

          System.out.println("Sorted List is");
      for(i=0;i<N;i++)
         System.out.println(x[i]);
   }
}
```

**Method 2**

```
import java.util.*;
class SELECTION
{   private double x[ ],large;
    private int N;
    public void getdata()
    {   Scanner s=new Scanner(System.in);
        System.out.println("Feed in the number of elements");
        N=s.nextInt();
            x=new double[N];
        System.out.println("Feed in the N elements");
        for(int i=0;i<N;i++)
           x[i]=s.nextDouble();
    }
      public void calculate()
     {  int pos;
        for(int i=N-1;i>0;i--) //where to put large & upto where to compare
        {   large=x[0];
           pos=0;         //ASSUMPTION
           for(int j=1;j<=i;j++)   //COMPARISON
           {   if (x[j]>large)
              {   large=x[j];
                 pos=j;
              }
           }
           x[pos]=x[i];
           x[i]=large;
            }
     }
    public void putdata()
    {   System.out.println("Sorted List is");
        for(int i=0;i<N;i++) System.out.println(x[i]);
    }
}
public class ex80_method2
{   public static void main(String args[ ])
    {  SELECTION X=new SELECTION();
        X.getdata();
        X.calculate();
        X.putdata();
    }
}
```

**PE81:** Write a program to read N number and arrange in descending order using selection sort

**Method 1**

```
import java.util.*;
public class ex81_method1
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      double x[ ],least;
      int N,pos,i,j;

      System.out.println("Feed in the number of elements");
      N=s.nextInt();
          x=new double[N];

      System.out.println("Feed in the N elements");
      for(i=0;i<N;i++)
      {
         x[i]=s.nextDouble();
      }

          for(i=N-1;i>0;i--) //where to put least & upto where to compare
      {   least=x[0];
         pos=0;        //ASSUMPTION
         for(j=1;j<=i;j++)   //COMPARISON
         {  if (x[j]<least)
            {   least=x[j];
               pos=j;
            }
         }
         x[pos]=x[i];
         x[i]=least;
      }
      System.out.println("Sorted List is");
      for(i=0;i<N;i++)
      {
         System.out.println(x[i]);
      }
   }
}
```

**Method 2**

```
import java.util.*;
class SELECTION
{  private double x[ ],least;
   private int N;
   public void getdata()
   {  Scanner s=new Scanner(System.in);
      System.out.println("Feed in the number of elements");
      N=s.nextInt();
      x=new double[N];
          System.out.println("Feed in the N elements");
      for(int i=0;i<N;i++)
         x[i]=s.nextDouble();
   }
   public void calculate()
   {  int pos;
      for(int i=N-1;i>0;i--) //where to put least & upto where to compare
      {  least=x[0];
         pos=0;        //ASSUMPTION
         for(int j=1;j<=i;j++)   //COMPARISON
         {  if (x[j]<least)
            {  least=x[j];
               pos=j;
            }
         }
         x[pos]=x[i];
         x[i]=least;
          }
       }
   public void putdata()
   {  System.out.println("Sorted List is");
      for(int i=0;i<N;i++) System.out.println(x[i]);
   }
}
public class ex81_method2
{  public static void main(String args[ ])
   {  SELECTION X=new SELECTION();
      X.getdata();
      X.calculate();
      X.putdata();
   }
}
```

**PE82:** Write a program to read N co-ordinates (x1,y1), (x2,y2),…… (xn,yn) and then calculates the length of the shortest line between any two coordinates and displays the two coordinates & the shortest length between them.

**Method 1**

```
import java.util.*;
public class ex82_method1
{
   public static void main(String args[ ])
   {
      double x[ ],y[ ],min=0;
      int N,pos,cpos,minpos1=0,minpos2=0,i;
      Scanner s=new Scanner(System.in);

      System.out.println("Feed in the number of points:");
      N=s.nextInt();
          x=new double[N];
      y=new double[N];

      System.out.println("Feed in the coordinates x & then y:");
      for(i=0;i<N;i++)
      {
         x[i]=s.nextDouble();
         y[i]=s.nextDouble();
      }
        for(pos=0;pos<=N-2;pos++)
        {
          for(cpos=pos+1;cpos<=N-1;cpos++)
          {
             if (pos==0 && cpos==1)
             {
                min=Math.sqrt((x[0]-x[1])*(x[0]-x[1])+(y[0]-y[1])*(y[0]-y[1]));
                minpos1=pos;
                minpos2=cpos;
             }

                else if(Math.sqrt((x[pos]-x[cpos])*(x[pos]-x[cpos])
                                        +(y[pos]-y[cpos])*(y[pos]-y[cpos]))<min)
             {
               min=Math.sqrt((x[pos]-x[cpos])*(x[pos]-x[cpos])
                                        +(y[pos]-y[cpos])*(y[pos]-y[cpos]));
                minpos1=pos;
                minpos2=cpos;
             }
```

```
        }
      }
        System.out.println("Minimum dist="+min+" and between ("+
        x[minpos1]+","+y[minpos1]+") and ("+x[minpos2]+","+ y[minpos2]+")");
   }
}
```

**Method 2**

```
import java.util.*;
class DIST
{  private double x[ ],y[ ],min;
   private int N,minpos1,minpos2;
   public void getdata()
   {  int i;
      Scanner s=new Scanner(System.in);
      System.out.println("Feed in the number of points:");
      N=s.nextInt();
      x=new double[N];
      y=new double[N];
          System.out.println("Feed in the coordinates x & then y:");
      for(i=0;i<N;i++)
      {
         x[i]=s.nextDouble();
         y[i]=s.nextDouble();
      }
   }
   public void calculate()
   {  int pos,cpos;
      for(pos=0;pos<=N-2;pos++)
      {  for(cpos=pos+1;cpos<=N-1;cpos++)
         {
            if (pos==0 && cpos==1)
            {
               min=Math.sqrt((x[0]-x[1])*(x[0]-x[1])+
                                                  (y[0]-y[1])*(y[0]-y[1]));
               minpos1=pos;
               minpos2=cpos;
            }
```

```
                    else  if(Math.sqrt((x[pos]-x[cpos])*(x[pos]-x[cpos])
                                +(y[pos]-y[cpos])*(y[pos]-y[cpos]))<min)
                    {
                            min=Math.sqrt((x[pos]-x[cpos])*(x[pos]-x[cpos])
                                    +(y[pos]-y[cpos])*(y[pos]-y[cpos]));
                             minpos1=pos;
                              minpos2=cpos;
                    }
            }
        }
    }

      public void putdata()
     {
          System.out.println("Minimum dist="+min+" and between
("+x[minpos1]+","+y[minpos1]+") and ("+x[minpos2]+","+ y[minpos2]+")");
      }
}

public class ex82_method2
{
    public static void main(String args[ ])
    {
       DIST X=new DIST();
       X.getdata();
       X.calculate();
       X.putdata();
    }
}
```

# Chapter 5

# 2D Numeric Arrays

**THE DIFFERENCE**

The main difference between 2D-arrays in C++ and arrays in Java is that arrays are treated like objects hence memory creation should be dynamic.

e.g. int x[ ][ ]=new int[10][10];

NOTE: dynamic array creation allows users to create arrays of user defined size hence eliminating the use of pointers

e.g. int x[ ][ ]=new int[m][n];

**Reading into an Array**

```
int x[ ][ ]=new int[10][10];
for(i=0;i<10;i=i+1)
{
        for(j=0;j<10;j=j+1)
                x[i][j]=s.nextInt();
}
```

NOTE: array size is 100. All are integers. First variable is called x[0][0] & the last is called x[9][9].

**Printing an Array**

```
for(i=0;i<10;i=i+1)
{
        for(j=0;j<10;j=j+1)
        {
                System.out.print(x[i][j]+"\t");
        }
        System.out.print("\n");   //or System.out.println();
}
```

**PE83:** Write a program to read MXN matrix and print it.

**PE84:** Write a program to read 2 matrices and add them

**Method 1**

```
import java.util.Scanner;
class ADD
{
   private int m,n;
   private double x[ ][ ],y[ ][ ],z[ ][ ];

   public void getdata()
   {
      Scanner s=new Scanner(System.in);
      int i,j;

      System.out.println("Feed in number of rows & columns of the matrix");
      m=s.nextInt();
      n=s.nextInt();

          x=new double[m][n];
      y=new double[m][n];
      z=new double[m][n];

      System.out.println("Feed in matrix1: ");
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
         {
            x[i][j]=s.nextDouble();
         }
      }
          System.out.println("Feed in matrix2: ");
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
         {
            y[i][j]=s.nextDouble();
         }
      }
   }
```

```
   public void calculate()
   {
      int i,j;
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
         {
            z[i][j]=x[i][j]+y[i][j];
         }
      }
   }

   public void putdata()
   {
      System.out.println("The Result is");
      int i,j;
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
         {
            System.out.print(z[i][j]+"\t");
         }
         System.out.println();
      }
   }
}

public class ex84_method1
{
   public static void main(String args[ ])
   {
      ADD X=new ADD();
      X.getdata();
      X.calculate();
      X.putdata();
   }
}
```

**Method 2**

```
import java.util.*;
public class ex84_method2
{
   public static void main(String args[ ])
   {
      int m,n,i,j;
      double x[ ][ ],y[ ][ ],z[ ][ ];

      Scanner s=new Scanner(System.in);

      System.out.println("Feed in number of rows & columns of the matrix");
      m=s.nextInt();
      n=s.nextInt();

         x=new double[m][n];
      y=new double[m][n];
      z=new double[m][n];

      System.out.println("Feed in matrix1: ");
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
         {
            x[i][j]=s.nextDouble();
         }
      }

         System.out.println("Feed in matrix2: ");
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
         {
            y[i][j]=s.nextDouble();
         }
      }
```

```
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
         {
            z[i][j]=x[i][j]+y[i][j];
         }
      }

          System.out.println("The Result is");
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
         {
            System.out.print(z[i][j]+"\t");
         }
         System.out.println();
      }
   }
}
```

**PE85:** Write a program to read 2 matrices and subtract them

**Method 1**

```
import java.util.Scanner;
class SUB
{
   private int m,n;
   private double x[ ][ ],y[ ][ ],z[ ][ ];

   public void getdata()
   {
      Scanner s=new Scanner(System.in);
      int i,j;

      System.out.println("Feed in number of rows & columns of the matrix");
      m=s.nextInt();
      n=s.nextInt();

         x=new double[m][n];
      y=new double[m][n];
      z=new double[m][n];

      System.out.println("Feed in matrix1: ");
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
         {
            x[i][j]=s.nextDouble();
         }
      }

          System.out.println("Feed in matrix2: ");
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
         {
            y[i][j]=s.nextDouble();
         }
      }
   }
```

```
   public void calculate()
   {
      int i,j;
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
         {
            z[i][j]=x[i][j]-y[i][j];
         }
      }
   }

   public void putdata()
   {
      System.out.println("The Result is");
      int i,j;
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
         {
            System.out.print(z[i][j]+"\t");
         }
         System.out.println();
      }
   }
}

public class ex85_method1
{
   public static void main(String args[ ])
   {
      SUB X=new SUB();
      X.getdata();
      X.calculate();
      X.putdata();
   }
}
```

**Method 2**

```
import java.util.*;
public class ex85_method2
{
   public static void main(String args[ ])
   {
      int m,n,i,j;
      double x[ ][ ],y[ ][ ],z[ ][ ];

      Scanner s=new Scanner(System.in);

      System.out.println("Feed in number of rows & columns of the matrix");
      m=s.nextInt();
      n=s.nextInt();

         x=new double[m][n];
      y=new double[m][n];
      z=new double[m][n];

      System.out.println("Feed in matrix1: ");
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
         {
            x[i][j]=s.nextDouble();
         }
      }

      System.out.println("Feed in matrix2: ");
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
         {
            y[i][j]=s.nextDouble();
         }
      }
```

```
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
         {
            z[i][j]=x[i][j]-y[i][j];
         }
      }

        System.out.println("The Result is");
       for(i=0;i<m;i++)
        {
         for(j=0;j<n;j++)
         {
            System.out.print(z[i][j]+"\t");
         }
         System.out.println();
      }
   }
}
```

**PE86:** Write a program to read 2 matrices and multiply them. Check if they are conformal for multiplication before actually multiplying them.

**Method 1**

```
import java.util.Scanner;
class MULT
{
   private int m,n,p,q;
   private double x[ ][ ],y[ ][ ],z[ ][ ];
   public void getdata()
   {
      Scanner s=new Scanner(System.in);
      int i,j,k;
       do
      {
      System.out.println("Feed in number of rows & columns of the matrix1");
      m=s.nextInt();
      n=s.nextInt();

      System.out.println("Feed in number of rows &columns of the matrix2");
          p=s.nextInt();
      q=s.nextInt();
      }while(n!=p);

       x=new double[m][n];
       y=new double[p][q];
       z=new double[m][q];

      System.out.println("Feed in matrix1: ");
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
            x[i][j]=s.nextDouble();
      }

          System.out.println("Feed in matrix2: ");
      for(j=0;j<n;j++)
      {
         for(k=0;k<q;k++)
            y[j][k]=s.nextDouble();
      }
   }
```

```
    public void calculate()
   {
      int i,j,k;
      for(i=0;i<m;i++)
      {
         for(k=0;k<q;k++)
         {
            z[i][k]=0; //NOT REQUIRED SINCE 2 CLASS METHOD
            for(j=0;j<n;j++)
            {
               z[i][k]+=x[i][j]*y[j][k];
            }
         }
      }
   }

   public void putdata()
   {
      System.out.println("The Result is");
      int i,k;
      for(i=0;i<m;i++)
      {
         for(k=0;k<q;k++)
         {
            System.out.print(z[i][k]+"\t");
         }
         System.out.println();
      }
   }
}

public class ex86_method1
{
   public static void main(String args[ ])
   {
      MULT X=new MULT();
      X.getdata();
      X.calculate();
      X.putdata();
   }
}
```

**Method 2**

```
import java.util.*;
public class ex86_method2
{
   public static void main(String args[ ])
   {
      int i,j,k,m,n,p,q;
      double x[ ][ ],y[ ][ ],z[ ][ ];

      Scanner s=new Scanner(System.in);
       do
       {
        System.out.println("Feed in number of rows &columns of the matrix1");
               m=s.nextInt();
                n=s.nextInt();

        System.out.println("Feed in number of rows &columns of the matrix2");
               p=s.nextInt();
               q=s.nextInt();
        }while(n!=p);

        x=new double[m][n];
        y=new double[p][q];
        z=new double[m][q];

      System.out.println("Feed in matrix1: ");
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
            x[i][j]=s.nextDouble();
      }
               System.out.println("Feed in matrix2: ");
      for(j=0;j<n;j++)
      {
         for(k=0;k<q;k++)
            y[j][k]=s.nextDouble();
       }
```

```
        for(i=0;i<m;i++)
        {
          for(k=0;k<q;k++)
          {
             z[i][k]=0;
             for(j=0;j<n;j++)
             {
                z[i][k]+=x[i][j]*y[j][k];
             }
          }
      }

        System.out.println("The Result is");
        for(i=0;i<m;i++)
        {
          for(k=0;k<q;k++)
          {
             System.out.print(z[i][k]+"\t");
          }
          System.out.println();
      }
   }
}
```

**PE87:** Write a program to read a matrix and find its transpose

```
import java.util.*;
class TRANS
{
   private int m,n;
   private double x[ ][ ],y[ ][ ];
   public void getdata()
   {
      Scanner s=new Scanner(System.in);
      int i,j;
      System.out.println("Feed in number of rows &
        columns of the matrix");
      m=s.nextInt();
      n=s.nextInt();

          x=new double[m][n];
      y=new double[n][m];

      System.out.println("Feed in matrix: ");
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
            x[i][j]=s.nextDouble();
      }
   }
   public void calculate()
   {
      int i,j;
      for(j=0;j<n;j++)
      {
         for(i=0;i<m;i++)
         {
            y[j][i]=x[i][j];
         }
      }
   }
```

```
   public void putdata()
   {
         int i,j;
      System.out.println("Transpose is ");
      for(j=0;j<n;j++)
      {
         for(i=0;i<m;i++)
         {
            System.out.print(y[j][i]+"\t");
         }
         System.out.println();
      }
   }
}
public class ex87_method1
{
   public static void main(String args[ ])
   {
      TRANS X=new TRANS();
      X.getdata();
      X.calculate();
      X.putdata();
   }
}
```

**PE88:** Write a program to read a matrix and find the highest and least element and their position

```
import java.util.*;
class MAXMIN
{
   private int m,n;
   private double x[ ][ ],max,min, maxposx,maxposy,minposx,minposy;

   public void getdata()
   {  Scanner s=new Scanner(System.in);
      int i,j;
          System.out.println("Feed in number of rows & columns of the matrix");
      m=s.nextInt();
      n=s.nextInt();

      x=new double[m][n];
      System.out.println("Feed in matrix: ");
      for(i=0;i<m;i++)
      {  for(j=0;j<n;j++)
            x[i][j]=s.nextDouble();
       }
   }
   public void calculate()
   {
      int i,j;
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
         {
            if (i==0 && j==0)
            {
               max=min=x[i][j];
               maxposx=maxposy=minposx=minposy=0;
            }
               else
               {
                     if (x[i][j]>max)
                     {
                           max=x[i][j];
                           maxposx=i;
                           maxposy=j;
                     }
```

```
                    if(x[i][j]<min)
                    {
                            min=x[i][j];
                            minposx=i;
                            minposy=j;
                }
              }
           }
    }
}

    public void putdata()
    {
       System.out.println("Maximum is "+max+" and at ("+maxposx
+","+maxposy+")");

      System.out.println("Minimum is "+min+" and at ("+minposx +","+minposy+")");
    }
}

public class ex88
{
    public static void main(String args[ ])
    {
       MAXMIN X=new MAXMIN();
       X.getdata();
       X.calculate();
       X.putdata();
    }
}
```

**PE89:** Write a program to read a matrix and check if it's an identity, upper triangular, lower triangular or neither.

```
import java.util.*;
class TYPE
{   private int m,n;
    private double x[ ][ ];
    public void getdata()
    {
        Scanner s=new Scanner(System.in);
        int i,j;
       do
        {
        System.out.println("Feed in number of rows & columns of the matrix");
            m=s.nextInt();
            n=s.nextInt();
        }while (m!=n);

        x=new double[m][n];
            System.out.println("Feed in matrix: ");
        for(i=0;i<m;i++)
        {
            for(j=0;j<n;j++)
            {
                x[i][j]=s.nextDouble();
            }
        }
    }
    public void putdata()
    {   int i,j;
        boolean identity=true;
        for(i=0;i<m;i++)
        {
            for(j=0;j<n;j++)
            {
                if (i!=j && x[i][j]!=0)
                    identity=false;
                if(i==j && x[i][j]!=1)
                    identity=false;
            }
        }
```

```
        if (identity)
         System.out.println("Matrix is an identity matrix");
        else
         System.out.println("Matrix is NOT an identity matrix");


       boolean upper=true;
       for(i=0;i<m;i++)
       {  for(j=0;j<i;j++)
          {  if (x[i][j]!=0)
                upper=false;
          }
       }
       if (upper)
              System.out.println("Matrix is an upper triangular matrix");
       else
              System.out.println("Matrix is NOT an upper triangular matrix");

        boolean lower=true;
        for(i=0;i<m;i++)
        {
                    for(j=i+1;j<n;j++)
          {
                          if (x[i][j]!=0)
                                lower=false;
          }
        }
             if (lower)
                    System.out.println("Matrix is an lower triangular matrix");
            else
             System.out.println("Matrix is NOT an lower triangular matrix");
   }
}

public class ex89
{  public static void main(String args[ ])
   {
         TYPE X=new TYPE();
      X.getdata();
      X.putdata();
   }
}
```

# Chapter 6

# Strings

**THE DIFFERENCE**

Character arrays was used in C++ to hold a string. It can yet be done in a similar way in JAVA. But JAVA provides a new class called String which can do the same.
The advantage is that Sting is a class, hence there would be many inbuilt functions to manipulate the 'string'.

**Reading into a String (OBJECT)**

We have learnt two ways of input

- BufferedReader
- Scanner

*By BufferedReader method*

```
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
String str;
str=br.readLine( );
```

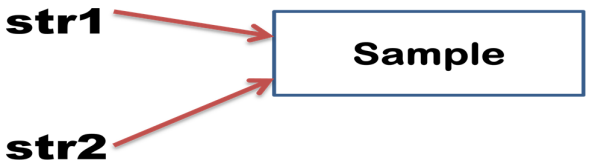
*By Scanner method*

```
Scanner s=new Scanner(System.in);
String str;
str=s.next( );
```

**Internal's of Strings**

As we know that objects are like pointers.
Hence when we do something like

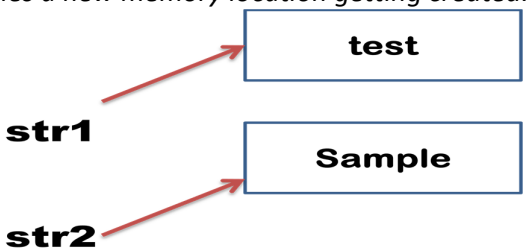
```
String str1,str2;
str1="Sample";
str2=str1;
```




Now if I do

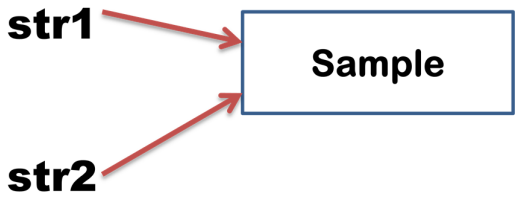
```
str1= "test";
```

*NOTE:*
*Assigning a new value to a string implies a new memory location getting created.*




NOTE:
Assigning a new value to a string implies a new memory location getting created.

Hence a new memory location is created (additional to the one already created) each time a new value is assigned to the string & the string variable (object) now points to this new string.
Hence a string can be thought of a fixed size memory to which we cannot add or remove anything. Doing so will create a new instance in memory & not change the original.
Hence it is not advisable to use Strings for changing 'strings' messages.
Conversely, doing
*String str1="Sample";*
*String str2="Sample";*
Will give the following




Hence, if we assign a string value to any String object, it first looks into the memory to see if this word already exists. If it does then the new object (pointer) would refer(point) to it & if it doesn't then it would create a new memory for it & the object would refer(point) to this memory [as it happened in case of str1].
Thus Strings can act like memory savers & even wasters (i.e. when you change the existing string a new memory is created for the same).
NOTE: Doing *str1==str2* would NOT compare the contents to which they point but instead will compare the locations to which they point.
*str1="Sample";*
*str2="Sample";*
Now str1==str2 will give 'true' (because they point to the same location).
whereas
*str1="Sample";*
*str2=new String("Sample");*
Now str1==str2 will give 'false' (because they do NOT point to the same location).
str2 now points to a 'new' location as compared to str1

**Content comparison in Strings**
*str1="Sample";*
*str2=new String("Sample");*
*str1.equals(str2)* will compare the contents to which str1 & str2 point. Now we will get a 'true'.
*str1.equals(str2)* is a case sensitive comparison (i.e. capital & small letters are not treated same). If you want a case insensitive comparison then we need to do
*str1.equalsIgnoreCase(str2);*

**String manipulation functions**

*.equals()*
*.equalsIgnoreCase()*
*.length()*
*.charAt()*
*.startsWith()*
*.endsWith()*
*.compareTo()*
*.concat()*
*.substring()*
*.replace()*
*.toLowerCase()*
*.toUpperCase()*

**Changing strings**

Changing strings is not a good practice as this creates new memory locations each time.
To handle this issue JAVA introduced a new class called StringBuffer.
Here changing the string actually changes the original & does not create a new instance.
Hence it is more space efficient.

**StringBuffer**

Creating a string buffer is like creating any other object
*StringBuffer sb=new StringBuffer(str);*
Where str is either a String object or a literal string in double quotes.
Manipulating the string within a buffer can be done using the in-built member functions of the class StringBuffer. They are……
*.reverse()*
*.append()*
*.insert()*
*.delete()*
*.deleteChatAt()*
*.replace()*

**PE90:** Write a program in JAVA showing the use of member functions of String & StringBuffer class & the interconversion between the two.

```
import java.io.*;
public class ex90
{
public static void main(String args[ ]) throws IOException
{
    BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
    String str,str2;

    System.out.println("Feed in a string");
    str=br.readLine();

    System.out.println("Length= "+str.length());

        System.out.println("First 3 letters are "+ str.substring(0,3));
    str=str+"java";
        System.out.println(str);

    System.out.println("4th character is "+ str.charAt(3));

    System.out.println("Feed in a string");
    str2=br.readLine();

    if(str.equalsIgnoreCase(str2))
         System.out.println("They are same");
    else
         System.out.println("They are not the same");

        if(str.startsWith("AB"))
        System.out.println("The string starts with Ab");
    if(str.endsWith("java"))
          System.out.println("The string ends in java");

    if(str.compareTo(str2)<0)
          System.out.println("String1 is smaller than string 2");
    else if(str.compareTo(str2)>0)
          System.out.println("String1 is larger than string 2");
    else
          System.out.println("Both are same");
         str2=str2+str;
```

```
    str2=str2.concat(str);
    System.out.println(str2);

    str=str.replace('a', 'c');
    System.out.println(str);

    str=str.toLowerCase();
    System.out.println(str);

    str=str.toUpperCase();
    System.out.println(str);

        StringBuffer sb=new StringBuffer(str);
    sb.reverse();
        System.out.println(sb);

    sb.append("test");
        System.out.println(sb);

        sb.insert(2,"hello");
        System.out.println(sb);

        sb.delete(2, 5);
        System.out.println(sb);

        sb.deleteCharAt(7);
        System.out.println(sb);

         sb.replace(5,6,"JAVA");
         System.out.println(sb);
 }
}
```

**PE91:** Write a program to read 2 words and check if they are same

```
import java.util.Scanner;
public class ex91
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      String str1,str2;

      System.out.println("Enter 2 strings:");
      str1=s.next();
      str2=s.next();

      if (str1.equalsIgnoreCase(str2))
      {
         System.out.println("Both strings are equal");
      }
      else
      {
         System.out.println("Both strings are NOT equal");
      }
   }
}
```

**PE92:** Write a program to read a word and check if it is a palindrome or not

```
import java.util.Scanner;
public class ex92
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      String str1,str2;

      System.out.println("Enter a strings:");
      str1=s.next();

         StringBuffer sb=new StringBuffer(str1);
      sb.reverse();
      str2=sb.toString();

      if (str1.equalsIgnoreCase(str2))
                  System.out.println("String is a pallindrome");
         else
               System.out.println("String is NOT a pallindrome");
   }
}
```

**PE93:** Write a program to read a word and replace every occurrence 'a' by 'j'.

```
import java.util.Scanner;
public class ex93
{
   public static void main(String args[ ])
   {  Scanner s=new Scanner(System.in);
      String str1;

      System.out.println("Enter a Word:");
      str1=s.next();

      str1=str1.replace('a','j');

      System.out.println("New string="+str1);
   }
}
```

**PE94:** Write a program to read any sentence & find its length, number of vowels, number of consonants, number of white spaces, number of capital letters, number of small letters, number of digits. Then display the sentence in upper case and in lower case.

```
import java.util.Scanner;
public class ex94
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      String str;
      int vowel=0,consonent=0,nosp=0,cap=0,small=0,nod=0;
      System.out.println("Enter a sentence");
      str=s.nextLine();

      int n=str.length();

         for(int i=0;i<n;i++)
      {
        if (str.charAt(i)=='a'||str.charAt(i)=='e'|| str.charAt(i)=='i' ||
                                        str.charAt(i)=='o'|| str.charAt(i)=='u'
                  ||str.charAt(i)=='A'||str.charAt(i)=='E'||str.charAt(i)=='I'
                                  ||str.charAt(i)=='O'||str.charAt(i)=='U')
        {
           vowel++;
        }
        else if((str.charAt(i)>='a'&& str.charAt(i)<='z')
              ||(str.charAt(i)>='A'&& str.charAt(i)<='Z'))
        {
           consonent++;
        }

       if (str.charAt(i)==' ')
                           nosp++;

        if (str.charAt(i)>='A' && str.charAt(i)<='Z')                cap++;

        if(str.charAt(i)>='a' && str.charAt(i)<='z')          small++;

        if(str.charAt(i)>='0' && str.charAt(i)<='9')          nod++;
         }//end of for

         System.out.println("Length="+n);
```

```
        System.out.println("Number of Vowels="+vowel);
        System.out.println("Number of Consonents="+consonent);
        System.out.println("Number of Spaces="+nosp);
        System.out.println("Number of Capitals="+cap);
        System.out.println("Number of Smalls="+small);
        System.out.println("Number of Digits="+nod);
        System.out.println("Upper Case="+ str.toUpperCase());
        System.out.println("Lower Case=" +str.toLowerCase());
    }
}
```

# Practice Questions

1. Write a function to find the norm of a matrix. The norm is defined as the square root of the sum of squares of all elements in the matrix.
2. Find the co-relation coefficient of a set of points (x,y), where correlation coefficient is given as

$$r = \frac{\sum xy - \sum x \sum y}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}}$$

3. A straight line y=ax+b can be fit to a set of points (x,y). Write a program that reads the x & y values & finds a & b are given as

$$a = \bar{y} - b\bar{x} \quad \text{and}$$

$$b = \frac{n\sum yx - \sum x \sum y}{[n\sum x^2 - (\sum x)^2]}$$

4. The **X** and **Y** coordinates of 10 different points are entered through the keyboard. Write a program to find the distance of last point from the first point (sum of distance between consecutive points)
5. Write a program that extracts part of the given string from the specified position. For example, if the sting is "Working with strings is fun", then if from position 4, 4 characters are to be extracted then the program should return string as "king". Moreover, if the position from where the string is to be extracted is given and the number of characters to be extracted is 0 then the program should extract entire string from the specified position
6. Write a program that replaces two or more consecutive blanks in a string by a single blank. For example, if the input is Grim return to the planet of apes!!
   **the output should be**
   Grim return to the planet of apes!!
7. Write a program to sort a set of names stored in an array in alphabetical order.
8. Write a program to count the number of occurrences of any two vowels in succession in a line of text.

9. Write a program to compute the mean, variance and standard deviation of a set of numbers using the following formula

$$\text{Mean} = 1/n \sum_{i=1}^{n} Xi$$

$$\text{Variance} = 1/n \sum_{i=1}^{n} (Xi-Xmean)^2$$

$$\text{Standared Deviation} = \sqrt{\text{Variance}}$$

10. **A minimax or saddle point in a two dimensional array is an element that is the minimum of its row and the maximum of its column, or vice verse. For example , in the following array A[I][J]**

**11 22 33 24**
**99 55 66 77**
**77 44 88 22**

**The element A[1][3]= 33 is a minimax**

**The element A[2][2]= 55 is a minimax**

**Write a program to identify and display all such saddle points and its location with in the two dimensional array, which is read from the keyboard**

11. Write a program to calculate the frequency of a given letter in a string.
12. Write a program to read a string & generate a frequncy table with the letter in one column & number of times it apeared in the other column.
13. Write a program to find all permutations of a string. (Find all possible combinations of the different characters of the string)
14. Decimal to binary conversion
15. Read a string & capitalize the first letter of every word
    Input: programming is fun
    Output: Programming Is Fun
16. Write a program to read a string & print it in alphabetical order. E.g. if the word if STRING the output should be GINRST
17. Read a 2D matrix and sort each row in an ascending order & print the new matrix
18. Write a program to interchange the words of a sentence
    Input: Insects eats apples
    Output: apples eats Insects

# Chapter 7

# Vectors

**Vectors v/s Arrays**

- Arrays can store only a fixed amount of data

***Vectors can store unlimited amount of data***

- Arrays can store only one type of data (e.g. int, double, etc….)

***Vectors can store only objects by can be of mixed types (e.g. Integer, Double, etc…)***

- You need to specify the array size while creating it

***You don't need to specify the vector capacity when creating it.***

**Restriction and its solution**

Note vector can store only objects not basic data types like int, double, etc…

*Hence we need to convert the int into Integer , double into Double, etc…*

**Conversion to Objects**

***int to Integer***

```
int x;
x=s.nextInt( );
Integer ox=new Integer(x);
```

**double to Double**

```
double x;
x=s.nextDouble( );
Double ox=new Double(x);
```

**Function of the Vector Class**

Step1: Create a vector

```
Vector v = new Vector( );
```

Step 2:Adding an object to a vector

```
v.addElement(ox);      //add at end
v.setElementAt(ox,position);
                                  //add at position
```

Step 3: Know the size of a vector

```
n=v.size( );
```

Step 4: Retrieve value for printing

```
System.out.println(v.get(position));
```

Step 5: Retrieve value for calculation

```
int data=Integer.parseInt(v.get(pos).toString());
```

**Inter – conversion from int to Integer & visa versa**

int ⇔ Integer

Integer ox=new Integer(x);

Vector array element ⇔ int

int x=*Integer.parseInt(v.get(pos).toString());*

**PE95:** Write a program in JAVA to sort a list of numbers using Vector object. Assume that the user hits a -1 to indicate end of data.

```
import java.util.Vector;
import java.util.Scanner;
public class vectors_ex95
{  public static void main(String args[ ])
  {    Scanner s=new Scanner(System.in);
       Vector v=new Vector();
       int x;
       while(true)
       {  System.out.println("Feed in an integer:-1 to quit)");
          x=s.nextInt();
          if (x==-1) break;
          Integer ox=new Integer(x); //converting int to Integer
          v.addElement(ox); //Only objects can be added to a vector
       }
                int n=v.size();
       System.out.println("You entered "+n+" elements");
       for(int pos=0;pos<=n-2;pos++)
       {  for(int cpos=pos+1;cpos<=n-1;cpos++)
          {
                  int pos_data=Integer.parseInt(v.get(pos).toString());
                  int cpos_data=Integer.parseInt(v.get(cpos).toString());
                  if (pos_data>cpos_data)
                      {
                              v.setElementAt(new Integer(cpos_data), pos);
                        v.setElementAt(new Integer(pos_data), cpos);
                      }
          }
       }
           System.out.println("Sorted List is");
       for(int i=0;i<n;i++)
          System.out.println(v.get(i));
   }
}
```

**PE96:** Write a program that accepts name & phone numbers into a VECTOR object. Assume that -1 for phone number indicates the end. Now sort this list based on name.

```
import java.util.Vector;
import java.util.Scanner;
public class vectors_ex96
{
   public static void main(String args[ ])
   {
      Scanner s=new Scanner(System.in);
      Vector v=new Vector();
      String phone,name;
       while(true)
       {
              System.out.println("Feed in the phone Number (-1 to quit)");
              phone=s.next();
              if (phone.equals("-1")) break;
              System.out.println("feed in the name");
              name=s.next();

              v.addElement(name);
              v.addElement(phone);
       }

         int n=v.size()/2;

      for(int pos=0;pos<=n-2;pos++)
      {
                for(int cpos=pos+1;cpos<=n-1;cpos++)
        {
                 String pos_data=v.get(pos*2).toString();
              String pos_extra_data=v.get(pos*2+1).toString();
              String cpos_data=v.get(cpos*2).toString();
              String cpos_extra_data=v.get(cpos*2+1).toString();
```

```
            if (pos_data.compareTo(cpos_data)>0)
            {     v.setElementAt(cpos_data, pos*2);
                  v.setElementAt(cpos_extra_data, pos*2+1);
                  v.setElementAt(pos_data, cpos*2);
                  v.setElementAt(pos_extra_data, cpos*2+1);
              }
           }
        }

        System.out.println("----------------------");
        System.out.println("Sorted List is");
        for(int i=0;i<n;i++)
        {
           System.out.println(v.get(2*i));
           System.out.println(v.get(2*i+1));
               System.out.println("--------------------");
        }
    }
}
```

**PE97:** Write a program in JAVA to sort a list of student data (name & 3 subject marks), based on total in descending order. Assume that the user hits a -1 to indicate end of data.

```
import java.util.Vector;
import java.util.Scanner;
public class vectors_ex97
{
   public static void main(String args[ ])
   {
         Scanner s=new Scanner(System.in);
      Vector v=new Vector();
      String name,choice;
         int m1,m2,m3;
      while(true)
      {
               System.out.println("feed in the name");
               name=s.next();
               System.out.println("feed in the 3 subject marks");
               m1=s.nextInt();
               m2=s.nextInt();
               m3=s.nextInt();
               v.addElement(name);
               v.addElement(new Integer(m1));
               v.addElement(new Integer(m2));
               v.addElement(new Integer(m3));
               System.out.println("Want to continue -1 to Quit");
               choice=s.next();
               if (choice.equals("-1")) break;
    }//end of while
               int n=v.size()/4;

          for(int pos=0;pos<=n-2;pos++)
      {
         for(int cpos=pos+1;cpos<=n-1;cpos++)
         {
            String pos_data=v.get(pos*4).toString();
               int pos_extra_data1=Integer.parseInt(v.get(pos*4+1).toString());
               int pos_extra_data2=Integer.parseInt(v.get(pos*4+2).toString());
                int pos_extra_data3=Integer.parseInt(v.get(pos*4+3).toString());
              int pos_total=pos_extra_data1+pos_extra_data2+pos_extra_data3;
               String cpos_data=v.get(cpos*4).toString();
              int cpos_extra_data1=Integer.parseInt(v.get(cpos*4+1).toString());
             int cpos_extra_data2=Integer.parseInt(v.get(cpos*4+2).toString());
```

```
        int cpos_extra_data3=Integer.parseInt(v.get(cpos*4+3).toString());
        int cpos_total=cpos_extra_data1+cpos_extra_data2+cpos_extra_data3;
        if (pos_total<cpos_total)
        {
                v.setElementAt(cpos_data, pos*4);
                v.setElementAt(new Integer(cpos_extra_data1), pos*4+1);
                v.setElementAt(new Integer(cpos_extra_data2), pos*4+2);
                v.setElementAt(new Integer(cpos_extra_data3), pos*4+3);

                 v.setElementAt(pos_data, cpos*4);
                v.setElementAt(new Integer(pos_extra_data1), cpos*4+1);
                v.setElementAt(new Integer(pos_extra_data2), cpos*4+2);
                v.setElementAt(new Integer(pos_extra_data3), cpos*4+3);
        }
     }
   }

        System.out.println("--------------------------");
    System.out.println("Sorted List is");
    for(int i=0;i<n;i++)
    {
                System.out.println(v.get(4*i));
                System.out.println(v.get(4*i+1));
                System.out.println(v.get(4*i+2));
                System.out.println(v.get(4*i+3));
                System.out.println("-----------------");
     }
   }
}
```

Other basic Vector class functions
.clear()
.lastElement()
.remove(……)
.removeAllElements()
.removeRange(…. , ….)
.toString()

**PE98:** Write a program in JAVA to demonstrate the use of the above methods.

**PE99:** Create 2 classes, Circle & Rectangle. Create user defined number of Circles & Rectangle objects. Find the largest Circle & Rectangle & the ratio of the two. Demonstrate Vectors as an heterogeneous storage space.

```
import java.util.Scanner;
import java.util.Vector;
class Circle
{
   private double r,area;
   public void getdata()
   {
      Scanner s=new Scanner(System.in);
      System.out.println("Enter radius:");
      r=s.nextDouble();
   }
   public double calculate()
   {
      area=Math.PI*r*r;
      return area;
   }
   public void putdata()
   {
      System.out.println("Area of Circle="+area);
   }
}
class Rectangle
{
   private double l,b,area;
   public void getdata()
   {
      Scanner s=new Scanner(System.in);
      System.out.println("Enter length & bredth:");
      l=s.nextDouble();
      b=s.nextDouble();
   }
   public double calculate()
   {
      area=l*b;
      return area;
   }
```

```
    public void putdata()
    {
        System.out.println("Area of Rectangle="+area);
    }
}
public class vector_ex99
{
    public static void main(String args[ ])
    {
        Vector v=new Vector();
        Circle C=new Circle();
        Rectangle R=new Rectangle();
        Scanner s=new Scanner(System.in);
        System.out.println("enter Number of Circles:");
        int noc=s.nextInt();
        System.out.println("enter Number of Rectangles:");
        int nor=s.nextInt();
            for(int i=0;i<noc;i++)
        {
            C.getdata();
            v.addElement(C);
        }

        for(int i=0;i<nor;i++)
        {
            R.getdata();
            v.addElement(R);
        }
            Circle max=new Circle();
        for(int i=0;i<noc;i++)
        {
            Circle obj=(Circle)v.get(i);

            if (i==0)
                max=obj;
            else
                if(obj.calculate()>max.calculate())
                    max=obj;
        }
```

```
      Rectangle maxr=new Rectangle();
      for(int i=noc;i<noc+nor;i++)
      {
         Rectangle obj=(Rectangle)v.get(i);

         if (i==0)
            maxr=obj;
         else
            if(obj.calculate()>maxr.calculate())
               maxr=obj;
      }
               System.out.print("Largest ");
               max.putdata();
        System.out.print("Largest ");
        maxr.putdata();
               if (max.calculate()>maxr.calculate())
                        System.out.println("Above Circle is the largest");
               if (maxr.calculate()>max.calculate())
                    System.out.println("Above Rectangle is the largest");
                 if (max.calculate()==maxr.calculate())
                        System.out.println("Both equal area");
   }
}
```

# Chapter 8

# Exceptions

Chapter 9

# Managing Errors and Exceptions

## 13.1 Introduction

Rarely does a program run successfully at its very first attempt. It is common to make mistakes while developing as well as typing a program. A mistake might lead to an error causing to program to produce unexpected results. *Errors* are the wrongs that can make a program go wrong.

An error may produce an incorrect output or may terminate the execution of the program abruptly or even may cause the system to crash. It is therefore important to detect and manage properly all the possible error conditions in the program so that the program will not terminate or crash during execution.

## 13.2 Types of Errors

Errors may broadly be classified into two categories:

- Compile-time errors
- Run-time errors

### Compile-Time Errors

All syntax errors will be detected and displayed by the Java compiler and therefore these errors are known as compile-time errors. Whenever the compiler displays an error, it will not create the **.class** file. It is therefore necessary that we fix all the errors before we can successfully compile and run the program.

**Program 13.1** ***Illustration of compile-time errors***

```
/* This program contains an error */
class Error1
{
        public static void main(String args[])
        {
            System.out.println("Hello Java!") // Missing;
        }
}
```

The Java compiler does a nice job of telling us where the errors are in the program. For example, if we have missed the semicolon at the end of print statement in Program 13.1, the following message will be displayed in the screen:

```
Error1.java :7: ';' expected
System.out.println ("Hello Java!")
^
1 error
```

We can now go to the appropriate line, correct the error, and recompile the program. Sometimes, a single error may be the source of multiple errors later in the compilation. For example, use of an undeclared variable in a number of places will cause a series of errors of type "undefined variable". We should generally consider the earliest errors as the major source of our problem. After we fix such an error, we should recompile the program and look for other errors.

Most of the compile-time errors are due to typing mistakes. Typographical errors are hard to find. We may have to check the code word by word, or even character by character. The most common problems are:

- Missing semicolons
- Missing (or mismatch of) brackets in classes and methods
- Misspelling of identifiers and keywords
- Missing double quotes in strings
- Use of undeclared variables
- Incompatible types in assignments / initialization
- Bad references to objects
- Use of = in place of = = operator
- And so on

Other errors we may encounter are related to directory paths. An error such as

```
javac : command not found
```

means that we have not set the path correctly. We must ensure that the path includes the directory where the Java executables are stored.

### Run-Time Errors

Sometimes, a program may compile successfully creating the **.class** file but may not run properly. Such programs may produce wrong results due to wrong logic or may terminate due to errors such as stack

overflow. Most common run-time errors are:

- Dividing an integer by zero
- Accessing an element that is out of the bounds of an array
- Trying to store a value into an array of an incompatible class or type
- Trying to cast an instance of a class to one of its subclasses
- Passing a parameter that is not in a valid range or value for a method
- Trying to illegally change the state of a thread
- Attempting to use a negative size for an array
- Using a null object reference as a legitimate object reference to access a method or a variable.
- Converting invalid string to a number
- Accessing a character that is out of bounds of a string
- And may more

When such errors are encountered, Java typically generates an error message and aborts the program. Program 13.2 illustrates how a run-time error causes termination of execution of the program.

**Program 13.2** ***Illustration of run-time errors***

```
class Error2
{
      public static void main(String args[ ])
      {
            int a = 10;
            int b = 5;
            int c = 5;
            int x = a/(b-c);        // Division by zero
            System.out.println("x = " + x);
            int y = a/(b+c);
            System.out.println("y = " + y);
      }
}
```

Program 13.2 is syntactically correct and therefore does not cause any problem during compilation. However, while executing, it displays the following message and stops without executing further statements.

```
java.lang.ArithmeticException: / by zero
     at Error2.main(Error2.java:10)
```

When Java run-time tries to execute a division by zero, it generates an error condition, which causes the program to stop after displaying an appropriate message.

## 13.3 Exceptions

An *exception* is a condition that is caused by a run-time error in the program. When the Java interpreter encounters an error such as dividing an integer by zero, it creates an exception object and throws it (i.e. informs us that an error has occurred).

If the exception object is not caught and handled properly, the interpreter will display an error message as shown in the output of Program 13.2 and will terminate the program. If we want the program to continue with the execution of the remaining code, then we should try to catch the exception object thrown by the error condition and then display an appropriate message for taking corrective actions. This task is known as *exception handling.*

The purpose of exception handling mechanism is to provide a means to detect and report an "exceptional circumstance" so that appropriate action can be taken. The mechanism suggests incorporation of a separate error handling code that performs the following tasks:

1. Find the problem (***Hit*** the exception).
2. Inform that an error has occurred (***Throw*** the exception)
3. Receive the error information (***Catch*** the exception)
4. Take corrective actions (***Handle*** the exception)

The error handling code basically consists of two segments, one to detect errors and to throw exceptions and the other to catch exceptions and to take appropriate actions.

When writing programs, we must always be on the lookout for places in the program where an exception could be generated. Some common exceptions that we must watch out for catching are listed in Table 13.1.

**Table 13.1 Common Java Exceptions**

| *Exception Type* | *Cause of Exception* |
|---|---|
| `ArithmeticException` | Caused by math errors such as division by zero |
| `ArrayIndexOutOfBoundsException` | Caused by bad array indexes |
| `ArrayStoreException` | Caused when a program tries to store the wrong type of data in an array |
| `FileNotFoundException` | Caused by an attempt to access a nonexistent file |
| `IOException` | Caused by general I/O failures, such as inability to read from a file |
| `NullPointerException` | Caused by referencing a null object |
| `NumberFormatException` | Caused when a conversion between strings and number fails |
| `OutOfMemoryException` | Caused when there's not enough memory to allocate a new object |
| `SecurityException` | Caused when an applet tries to perform an action not allowed by the browser's security setting |
| `StackOverFlowException` | Caused when the system runs out of stack space |
| `StringIndexOutOfBoundsException` | Caused when a program attempts to access a nonexistent character position in a string |

## 13.4 Syntax of Exception Handling Code

The basic concepts of exception handling are throwing an exception and catching it. This is illustrated in Fig. 13.1.

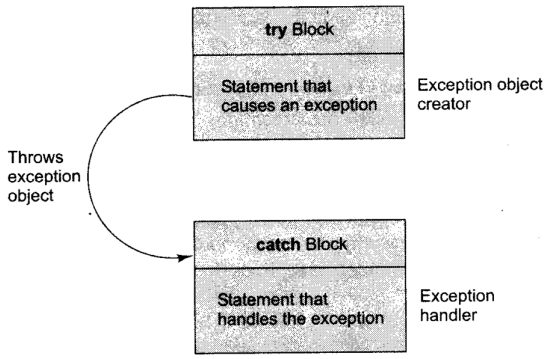



**Fig. 13.1** *Exception handling mechanism*

Java uses a keyword **try** to preface a block of code that is likely to cause an error condition and "throw" an exception. A catch block defined by the keyword **catch** "catches" the exception "thrown" by thee try block and handles it appropriately. The catch block is added immediately after the try block. The following example illustrates the use of simple **try** and **catch** statements:

```
.....
.....
try
{
    statement;      // generates an exception
}
catch (Exception-type e)
{
    statement;      // processes the exception
}
.....
.....
```

The try block can have one or more statements that could generate an exception. If any one statement generates an exception, the remaining statements in the block are skipped and execution jumps to the catch block that is placed next to the try block.

The catch block too can have one or more statements that are necessary to process the exception. Remember that every **try** statement should be followed by *at least one* **catch** statement; otherwise compilation error will occur.

Note that the **catch** statement works like a method definition. The **catch** statement is passed a single parameter, which is reference to the exception object thrown (by the try block). If the catch parameter matches with the type of exception object, then the exception is caught and statements in the catch block will be executed. Otherwise, the exception is not caught and the default exception handler will cause the execution to terminate.

Program 13.3 illustrates the use of try and catch blocks to handle an arithmetic exception. Note that Program 13.3 is a modified version of Program 13.2.

**Program 13.3** ***Using try and catch for exception handling***

```
class Error3
{
        public static void main(String args[ ])
        {
                int a = 10;
                int b = 5;
                int c = 5;
                int x, y ;
                try
                {
                        x = a / (b-c);          // Exception here
                }
                catch (ArithmeticException e)
                {
                        System.out.println("Division by zero");
                }
                y = a / (b+c);
                System.out.println("y = " + y);
        }
}
```

Program 13.3 displays the following output:

```
Division by zero
y = 1
```

Note that the program did not stop at the point of exceptional condition. It catches the error condition, prints the error message, and then continues the execution, as if nothing has happened. Compare with the output of Program 13.2 which did not give the value of **y**.

Program 13.4 shows another example of using exception handling mechanism. Here, the try-catch block catches the invalid entries in the list of command line arguments.

**Program** *Catching invalid command line arguments*

```
class CLineInput
{
    public static void main(String args[ ])
    {
        int invalid = 0;    // Number of invalid arguments
        int number, count = 0;
        for (int i = 0; i < args.length; i++)
        {
            try
            {
                number = Integer.parseInt(args[i]);
            }
            catch(NumberFormatException e)
            {
              invalid = invalid + 1;        // Caught an invalid number
              System.out.println("Invalid Number: "+args[i]);
              continue;                     // Skip the remaining part of the loop
            }
            count = count + 1;
        }
        System.out.println("Valid Numbers = " + count);
        System.out.println("Invalid Numbers = " + invalid);
    }
}
```

Note the use of the wrapper class **Integer** to obtain an **int** number from a string:

```
number = Integer.parseInt(args[i])
```

Remember that the numbers are supplied to the program through the command line and therefore they are stored as strings in the array **args[ ]**. Since the above statement is placed in the try block, an exception is thrown if the string is improperly formatted and the number is not included in the count.

When we run the program with the command line:

```
java CLineInput 15 25.75 40 Java 10.5 65
```

it produces the following output:

```
Invalid Number:  25.75
Invalid Number: Java
Invalid Number: 10.5
Valid Numbers   = 3
Invalid Numbers = 3
```

## 13.5 Multiple Catch Statements

It is possible to have more than one catch statement in the catch block as illustrated below:

```
.....
.....
try
{
    statement ;              // generates an exception
}
catch (Exception-Type-1 e)
{
    statement;               // processes exception type 1
}
catch (Exception-Type-2 e)
{
    statement;               // processes exception type 2
}
.
.
.
catch (Exception-Type-N e)
{
    statement ;              // processes exception type N
 }
.....
.....
```

When an exception in a **try** block is generated, the Java treats the multiple **catch** statements like cases in a **switch** statement. The first statement whose parameter matches with the exception object will be executed, and the remaining statements will skipped.

Note that Java does not require any processing of the exception at all. We can simply have a catch statement with an empty block to avoid program abortion.

Example:

```
catch (Exception e);
```

The **catch** statement simply ends with a semicolon, which does nothing. This statement will catch an exception and then ignore it.

**Program** ***Using multiple catch blocks***

```
class Erro4
{
      public static void main(String args[ ])
      {
            int a[ ] = {5, 10};
            int b = 5;
            try
            {
                  int x = a[2] / b - a[1];
            }
```

*(Continued)*

**Program** *(Continued)*

```
            catch(ArithmeticException e)
            {
                  System.out.println("Division by zero");
            }
            catch(ArrayIndexOutOfBoundsException e)
            {
                  System.out.println("Array index error");
            }
            catch(ArrayStoreException e)
            {
                  System.out.println("Wrong data type");
            }
            int y = a[1]./ a[0];
            System.out.println("y = " + y);
      }
}
```

Program 13.5 uses a chain of catch blocks and, when run, produces the following output:

```
Array index error
y = 2
```

Note that the array element a[2] does not exist because array **a** is defined to have only two elements, a[0] and a[1]. Therefore, the index 2 is outside the array boundary thus causing the block

```
Catch(ArrayIndexOutOfBoundsException e)
```

to catch and handle the error. Remaining catch blocks are skipped.

## 13.6 Using Finally Statement

Java supports another statement known as **finally** statement that can be used to handle an exception that is not caught by any of the previous catch statements. **finally** block can be used to handle any exception generated within a try block. It may be added immediately after the try block or after the last catch block shown as follows:

```
try
{
    ..........
    ..........
}
finally
{
    ..........
    ..........
}
```

```
try
{
    ..........
    ..........
}
catch   (....)
{
    ..........
    ..........
}
```

```
catch     (....)
{
          ..........
          ..........
}
.
.
.
.
finally
{
          ..........
          ..........
}
```

When a **finally** block is defined, this is guaranteed to execute, regardless of whether or not in exception is thrown. As a result, we can use it to perform certain house-keeping operations such as closing files and releasing system resources.

In Program 13.5, we may include the last two statements inside a **finally** block as shown below:

```
finally
{
     int y = a[1]/a[0];
     System.out.println("y = " +y);
}
```

This will produce the same output.

## 13.7 Throwing Our Own Exceptions

There may be times when we would like to throw our own exceptions. We can do this by using the keyword **throw** as follows:

```
throw new Thorwable_subclass;
```

Examples:

```
throw new ArithmeticException( );
throw new NumberFormatException( );
```

Program 13.6 demonstrates the use of a user-defined subclass of Throwable class. Note that **Exception** is a subclass of **Throwable** and therefore **MyException** is a subclass of **Throwable** class. An object of a class that extends **Throwable** can be thrown and caught.

**Program 13.6** ***Throwing our own exception***

```
import java.lang.Exception;
class MyException extends Exception
```

(*Continued*)

**Program** *(Continued)*

```
{
        MyException(String message)
        {
                super(message);
        }
}
class TestMyException
{
        public static void main(Strings args[ ])
        {
                int x = 5, y = 1000;
                try
                {
                    float z = (float) x / (float) y ;
                    if(z.< 0.01)
                    {
                        throw new MyException("Number is too small");
                    }
                }
                catch (MyException e)
                {
                    System.out.println("Caught my exception");
                    System.out.println(e.getMessage( ) );
                }
                finally
                {
                    System.out.println("I am always here");
                }
        }
}
```

A run of program 13.6 produces:

```
Caught my exception
Number is too small
I am always here
```

The object e which contains the error message "Number is too small" is caught by the **catch** block which then displays the message using the getMessage( ) method.

Note that Program 13.6 also illustrates the use of **finally** block. The last line of output is produced by the **finally** block.

# Chapter 9

# Inheritance

**Inheritance in Java**

- Setting up single inheritance is as follows

```
class Parent
{
        …….
}
class Child extends Parent
{
        …….
}
```

- Note extends is used by the child class to indicate it is an extension of which class.
- Where is the visibility mode???
- Will the inherited members come in private/public sections of the Child class???
- Just like C++ private does not inherit.
- public will inherit into public
- protected will inherit into protected
- NOTE no mode is NOT private mode, but it is like public mode in JAVA.
- protected is ACCESSIBLE from outside the class
- What's the difference in the various mode then?

| | Public | protected | No mode | private protected | private |
|---|---|---|---|---|---|
| Same class | Yes | Yes | Yes | Yes | Yes |
| Subclass in same package | Yes | Yes | Yes | Yes | No |
| Other class same package (outsider) | Yes | Yes | Yes | No | No |
| Subclass other packages | Yes | Yes | No | Yes | No |
| Non-subclass (outsider) other packages | Yes | No | No | No | No |

- See public, protected, no-mode columns closely

**PE100:** Write a program in JAVA to create a class employee which maintains the name, id & salary of the employee. Create another class manager which additionally maintains the designation of that manager. Accept details of two managers & print the details of the one with a higher salary.

**Method 1**

```
import java.util.*;
class Employee
{
   private String name;
   private int id;
   protected double sal;
   public void getdatae()
   {  Scanner s=new Scanner(System.in);
      System.out.println("Enter name, id, salary:");
      name=s.next();
      id=s.nextInt();
      sal=s.nextDouble();
   }
   public void putdatae()
   {
      System.out.println("Name="+name);
      System.out.println("ID="+id);
      System.out.println("Salary="+sal);
   }
}
class Manager extends Employee
{
   private String desig;

   public void getdatam()
   {
      getdatae();
      Scanner s=new Scanner(System.in);
      System.out.println("Enter designation");
      desig=s.nextLine();
   }
   public  void putdatam()
   {  putdatae();
      System.out.println("Designation="+desig);
   }
```

```
   public void checksal(Manager B)
   {
      if (sal>B.sal)   putdatam();
      else if (B.sal>sal)  B.putdatam();
      else
      {
               putdatam();
               B.putdatam();
      }
   }
}
public class ex100
{
   public static void main(String args[ ])
   {
      Manager X=new Manager();
      Manager Y=new Manager();
      X.getdatam();
      Y.getdatam();
      X.checksal(Y);
   }
}
```

**Method 2**

```
import java.util.*;
class Employee
{
   private String name;
   private int id;
   protected double sal;
   public void getdata()
   {  Scanner s=new Scanner(System.in);
      System.out.println("Enter name, id, salary:");
      name=s.next();
      id=s.nextInt();
      sal=s.nextDouble();
   }

   public void putdata()
   {
      System.out.println("Name="+name);
      System.out.println("ID="+id);
      System.out.println("Salary="+sal);
   }
}

class Manager extends Employee
{
   private String desig;

   public void getdata()
   {
      super.getdata();
      Scanner s=new Scanner(System.in);
      System.out.println("Enter designation");
      desig=s.nextLine();
   }

   public void putdata()
   {  super.putdata();
      System.out.println("Designation="+desig);
   }
```

```
   public void checksal(Manager B)
   {
              if (sal>B.sal)   super.putdata();
        else  if (B.sal>sal)  B.putdata();
        else
         {
                    super.putdata();
           B.putdata();
         }
   }
}

public class ex100_method
{
   public static void main(String args[ ])
   {
      Manager X=new Manager();
      Manager Y=new Manager();
      X.getdata();
      Y.getdata();
      X.checksal(Y);
   }
}
```

**Constructors in Inheritance in Java**

- Constructors can be explicitly called as follows:

```
class A
{  private int x;
   public A( )
   {
      x=100;
   }
   public void putdataA( )
   {
      System.out.println("x="+x);
   }
}
class B extends A
{  private int y;
   public B( )
   {  super( );              //calls the base class constructor
                             //not required when its default constructor
      y=5;
   }
   public void putdataB()
   {
      putdataA();
      System.out.println("y="+y);
   }
}
public class ex_const_1
{
   public static void main(String args[ ])
   {
      B X=new B();
      X.putdataB();
   }
}
```

OUTPUT:
x=100
y=5

**Parameterized Constructors in Inheritance in Java**

```
class A
{
   private int x;
   public A(int m)
   {
      x=m;
   }
   public void putdataA()
   {
      System.out.println("x="+x);
   }
}
class B extends A
{
   private int y;
   public B(int a, int b)
   {
      super(a);       //necessary in this case, calls base class constr.
      y=b;
   }
   public void putdataB()
   {
      putdataA();
      System.out.println("y="+y);
   }
}
public class ex_const_2
{
   public static void main(String args[ ])
   {
      B X=new B(10,5);
      X.putdataB();
   }
}
```

**PE101:** Write a program in JAVA to create a class ‘ citizen’ which maintains the name, income & PAN number. Create another derived class ‘ employee’ which contains the employee number & salary (income ≥ salary). Create another class ‘ manager’ derived from employee, which maintains his/her designation. Test the program for a single manager.

**PE102:** Write a program in JAVA to demonstrate default constructors in multilevel inheritance.

**PE103:** Write a program in JAVA to demonstrate parameterized constructors in multilevel inheritance.

**PE104:** Write a program in JAVA for ABC Ltd. which has the following categories of employees – managers,scientists & laborers. Name & employee number are to be maintained for all the above types employees. In addition to this the designation of managers & number of publications of the scientists are too be maintained. Test the above functionality with 10 managers,10 scientists & 100 laborers.

**PE105:** Write a program in JAVA to create a class ‘ student’ that maintains the name & roll number of the student. Derive a class ‘ marks’ that maintains the five subject marks of the student. Create another base class called ‘ sports’ which maintains the sports score. Derive a class ‘ results’ which calculates the result of the student as follows: Final result = (Sum of 5 subject marks + 5% of the sports score)/5;

**Containership, class within class, nested classes**

```
class B
{
        ……
        ……
};
class A
{
        B X;            //X is an object of class B
        ……..
        ……
};
```

**PE106:** Write a program in JAVA to create a class ' student' containing his/her name & roll number. Additionally create 2 classes for his/her own marks & sports points. Use containership to accept & print the marks & sports. Calculate the final result as
Final result = (Sum of 5 subject marks + 5% of sports points)/5;
Accept the details for 10 students & print their final result too.

```
import java.util.*;
class student
{
   private String name;
   private int roll;
   private marks M;
   private sports S;
   private double result;

   public void getdata()
   {
      Scanner s=new Scanner(System.in);
      System.out.println("Enter the name & roll      number");
      name=s.next();
      roll=s.nextInt();
      M=new marks();
      M.getdata();
      S=new sports();
      S.getdata();
   }

   public void calculate()
   {
      result=(M.getsum()+0.05*S.getpts())/5;
   }

   public void putdata()
   {
      System.out.println("Name="+name);
      System.out.println("Roll Number="+roll);
      System.out.println("Result="+result);
      System.out.println("-----------------");
   }
}
```

```
class marks
{    private int m1,m2,m3,m4,m5;
   public void getdata()
   {   Scanner s=new Scanner(System.in);
       System.out.println("Enter the 5 subject marks");
       m1=s.nextInt();
       m2=s.nextInt();
       m3=s.nextInt();
       m4=s.nextInt();
       m5=s.nextInt();
   }
   public int getsum()
   {
       return(m1+m2+m3+m4+m5);
   }
}
class sports
{   private double pts;
   public void getdata()
   {
       Scanner s=new Scanner(System.in);
       System.out.println("Enter the sports points");
       pts=s.nextDouble();
   }
   public double getpts()
   {
       return pts;
   }
}
public class ex106
{   public static void main(String args[ ])
   {
       student X[ ]=new student[10];
       for (int i=0;i<10;i++)
       {   X[i]=new student();
           X[i].getdata();
           X[i].calculate();
           X[i].putdata();
       }
   }
}
```

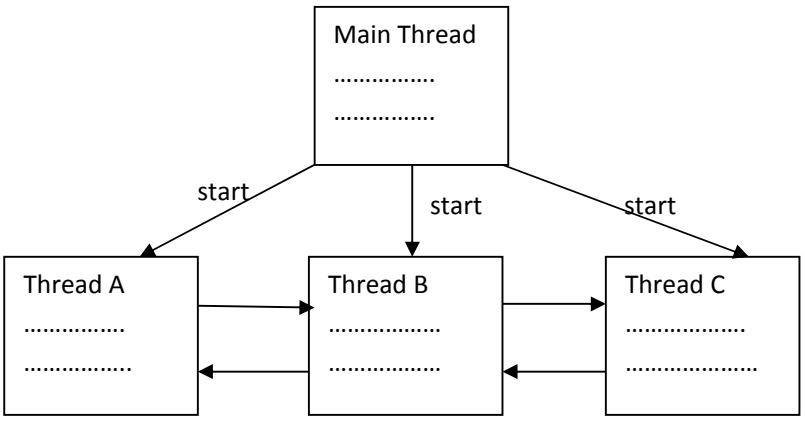
# Chapter 10

# MultiThreading

Introduction

Imagine you are sitting on your computer & recording a DVD, while playing music & also chatting online with your friends. You can also see your computer clock ticking away minute after minute. You do get a feeling that all this is happening simultaneously. But actually we have only one processor in our computer & one processor can process only one thing at one time. Thus when the computer is recording the DVD, all the other thinks are on hold. It is obviously not appearing so to you. Let's see why? The computer runs all the programs partly in a round robin fashion for small amounts of time. Thus, the DVD recording happens for a small amount of time & at the same time all the other things are waiting. Then, before completing, the recording is stopped & the next program (process) is picked up, say the music, & it is run for some small amount of time; and so on. Then, the whole cycle keeps on repeating till the programs complete. The repeat chance of an program (process), if it comes in less that 0.1 seconds, then our human brain will not be able to notice the break in the process. That's how we get a feeling of all running concurrently. This is also called multitasking.

Let's take another example, of an internet browser. Here we may be having multiple pages open. May be one of the pages may be your mail inbox, other may be the latest cricket scores, other may be the lates stock market indices, one may be a streaming video, etc. These all pages are part of one software, say Google Chrome, but all are appearing to be simultaneously refeshing with live information. The processor again being single is picking one page at a time & refreshing it perodically, though it appears simultanous to us. This is also called as multithreading, with each window being called a thread. Let's see a bit more technicalities in the above discussion? The main software or lets say the may program of the browser is called the main thread & it deligates the job of loading each new web address in a new window to a new thread. Thus we have a main thread with fork's new child thread processes & deligates work to them. Though not very evedent in this case but the threads can intercommunicate between each other, sharing resources that are available. The main thread can also stop, suspend, resume, etc, any of its child threads. The following diagram shows a basic view of this discussion.



Examples:

Lets get into our first example & understand the process of threading.

Here we will create 2 additional (child) threads, one for printing 'a' to 'z' and the other for printing 1 to 26, each 50 times.

Each thread is an independent unit of code (actually a class) having its own main (actually, it has a function called run(), that acts like a main). The main() creates the object of those classes & activates the run() function (by calling the start() function). Sounds confusing? Let's see it in detail:

```
class letters extends Thread
{
   public void run()
   {
      for(int times=1;times<=50;times++)
      for(int i='a';i<='z';i++)
         System.out.println((char)i);
   }
}
class nos extends Thread
{
   public void run()
   {
      for(int times=1;times<=50;times++)
      for(int i=1;i<=26;i++)
         System.out.println(i);
   }
}
public class ex1
{
   public static void main(String args[ ])
   {
      letters L=new letters();
      nos N=new nos();
      L.start();
      N.start();
   }
}
```

As can be seen above, we have created 2 class, letters and nos, for printing 'a' to 'z' 50 times & 1 to 26, 50 times. Firstly, each of the 2 classes are derives classes (sub-classes) of the class Thread. The Thread class in an inbuilt class of the lang package. The run() function which has been written in both the classes overrides the Thread run() function. In the

main(), its a two step process – create object of each class & call the start function of each class. Note start() function is a function of the Thread class & due to inheritance, it is also part of the derived classes. Hence, we are able to call the start() function using the derived class object. The start() function inturn calls the run() function.

In a normal program, not involving threading, the main() executes line by line & at each function call, the control transfers to the function & on completion of the function the control returns back to main() & then the main() continues execution. Hence, the programming control is either in the main() or in the function. With threading, when the main() starts a thread, say L.start(), what it does, is that, it kind off creates a new programming control for the run() function. Since we have only one processor, the main() when is executing will create the new programming control for the run() function of the letter class, but not necessary that the new programming control will start executing at that time. It could & in that case the main() would pause or it could be that the main() would continue & thus creating a new programming control for the mos run() function too. Even when the run() of say the letter class is being executed by the process, its does not necessarily run up to completion. It could partly run & then the processor would pause it & go for the other programming control, which too it could run for some time & with over to the previous programming control. Who is going to be executed, when & for how long is unpredictable. Hence the output will be garbelled with intemingling of the 'a' to 'z' & 1 to 26, each 50 times in any other. More so, the order witll be different at each execution.

What is the advantage of what we just did? In fact the major advantage of multithreading is reduced waiting time. No one is waiting for the other one to complete. Each one gets there small time quota to be executed by the processor & then they have to wait for a short time, while the processor is working with the other thread(s), and then its chance will come again.

Now let's try to understand the terms 'join' and 'isAlive'. When start() is executed by main() the thread 'isAlive' and stays alive till the run() function hasn't finished. So even the processor is executing another thread, yet the theads not with with processor 'isAlive'. When the run() function of a thread completes, the thread will be termed 'dead' and is said to 'join' main(), who created it.

Let's understand the above concept with the next example.

Here we will print 'a' to 'z' and 1 to 26 using multithreading, but in such a way that all 'a' to 'z' should be printed first & then only should 1 to 26 come. Note we are expected to make the unpredictable output of multithreading into a predictable pattern.

```
class letters extends Thread
{
   public void run()
   {
      for(int i='a';i<='z';i++)
         System.out.println((char)i);
   }
}

class nos extends Thread
{
   public void run()
   {
      for(int i=1;i<=26;i++)
         System.out.println(i);
   }
}

public class ex4
{
   public static void main(String args[ ])
   {
      letters A=new letters();
      nos B=new nos();

      A.start();
      try
      {
         A.join();            //waiting for A to join main()
      }
      catch(Exception e){}

      B.start();
   }
}
```

```
/*Alternate SLOW TECHNIQUE*/
class letters extends Thread
{
   public void run()
   {
      for(int i='a';i<='z';i++)
         System.out.println((char)i);
   }
}
class nos extends Thread
{
   public void run()
   {
      for(int i=1;i<=26;i++)
         System.out.println(i);
   }
}
public class ex5
{
   public static void main(String args[ ])
   {
      letters A=new letters();
      nos B=new nos ();

      A.start();
      do
      {
      }while(A.isAlive());   //If A's alive, go back to 'do'

      B.start();
   }
}
```

As can be seen from the main() above, we have ensured that we do not execute B.start() till we are sure A is dead. Obviouly, we are killing the advantage of multithreading, as now it flow of the programs looks to be sequential. B will start only after A's dead.

If you think a bit, you will realize that the above program objective could be easily achieved without multithreading. main() would first call a function that prints 'a' to 'z' and then main() will call another function that will print 1 to 26. Thus, killing multithreading really is not advantageous as there are other ways of doing the same program.

By now you must have got an impression that every child Thread is a new class. That's not completely true. Every child thread should be a new object, not necessarily a new class. Let's see the following example:

Create 2 Threads, each printing from 1 to 1000

Think about it.... If we create 2 class, both will have the same run() function code. We need one class & 2 objects, hence each will have its own run().

```
class test extends Thread
{
   public void run()
   {      for(int i=1;i<=1000;i++)
             System.out.println("I am a thread. Now I am printing "+i);
   }
}
public class ex2
{  public static void main(String args[ ])
   {
      test X=new test();   //Two objects of the same class
      test Y=new test();
      X.start();
      Y.start();
   }
}
```

Thus, the number of class does not depend on the number of child threads, but on the number of distinct actions the child threads need. Like in the example above we have 2 child threads but one type of action, wheras in the previous example we had two child threads & two distinct actions.

Let's try to get 1 to 1000 in an ordered way now.

First one child thread prints 1 to 1000 & then only the other one should start.

```
class test extends Thread
{
   public void run()
   {
      for(int i=1;i<=1000;i++)
      {
         System.out.println(i);
      }
   }
}
```

```
public class ex4
{
   public static void main(String args[ ])
   {
      test A=new test();
      test B=new test();

      A.start();
      try
      {
         A.join();
      }
      catch(Exception e){}

      B.start();
   }
}

/*Alternate SLOW TECHNIQUE*/
class test extends Thread
{  public void run()
   {
      for(int i=1;i<=1000;i++)
      {
         System.out.println(i);
      }
   }
}
public class ex5
{   public static void main(String args[ ])
   {  test A=new test();
      test B=new test();

      A.start();
      do
      {
      }while(A.isAlive());

      B.start();
   }
}
```

/* Let's see another way to achieve the same. */

```
class display
{
   public synchronized void putdata()
   {
      for(int i=1;i<=1000;i++)
      {
         System.out.println(i);
      }
   }
}
class test extends Thread
{
   display D;
   public test(display m)
   {
      D=m;
      Thread t=new Thread(this);
      t.start();
   }

   public void run()
   {
      D.putdata();
   }
}
public class ex8
{
   public static void main(String args[ ])
   {
      display x=new display();

      test A=new test(x);
      test B=new test(x);

   }
}
```

Here we notice two major changes. Firstly, in main we created the 2 thread objects & called parameterized constructor with a common object 'x'. The parameterized constructor accepts the argument & sets it as its private entity. Thus, A and B but have their private variable D set to the same object 'x'. Then we convert 'this' object (i.e. A or B) into a thread

& start the thread. Th moment the thread starts it enters run() & run() calls putdata(). Thus we expect that putdata() for say A is running & printing 1 to 1000, when suddenly, in between, the processor may decide to 'yield' A & take control of thread B. Thus B too, would try to enter putdata(). But since putdata() is synchronized, only one thread can be in it at one time. Its like saying the A had put a lock when it enters the putdata() function & till A does not complete putdata(), the lock will not be released. Thus, when B tries to enter putdata() it will have to wait as it is locked by A. The thread B will idle for some time & then the processor will yield B & transfer control back to A & continue from where it left off. Eventually, after repeated transfer from A to B & vice versa, A would complete its putdata() & the lock would release & then when the processor goes to B, B will get its chance to enter putdata(). Thus the output would be 1 to 1000 for A & then 1 to 1000 for B.

### Blocking a Thread

A thread can also be temporarily suspended or blocked from entering into the runnable and subsequently running state by using either of the following thread methods:

```
sleep( )          // blocked for a specified time
suspend( )        // blocked until further orders
wait( )           // blocked until certain condition occurs
```

These methods cause the thread to go into the ***blocked*** (or ***not-runnable***) state. The thread will return to the runnable state when the specified time is elapsed in the case of **sleep( ),** the **resume( )** method is invoked in the case of **suspend( ),** and the **notify( )** method is called in the case of **wait( ).**

## 12.5 Life Cycle of a Thread

During the life time of a thread, there are many states it can enter. They include:

1. Newborn state
2. Runnable state
3. Running state
4. Blocked state
5. Dead state

A thread is always in one of these five states. It can move from one state to another via a variety of ways as shown in Fig. 12.3.

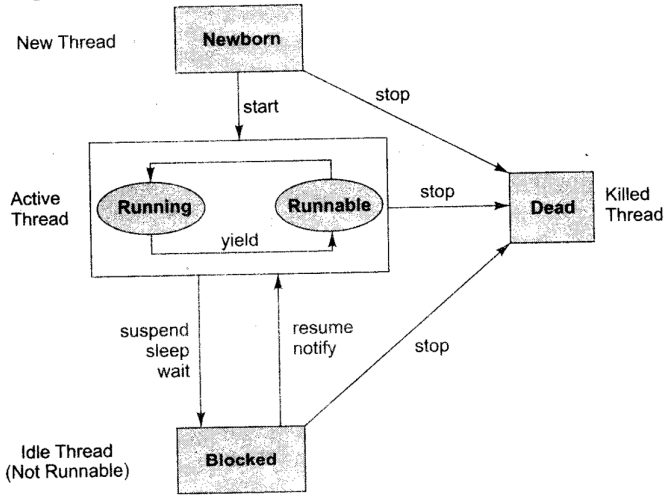



**Fig. 12.3** *State transition diagram of a thread*

### Newborn State

When we create a thread object, the thread is born and is said to be in *newborn* state. The thread is not yet scheduled for running. At this state, we can do only one of the following things with it:

- Schedule it for running using **start( )** method.
- Kill it using **stop( )** method.

If scheduled, it moves to the runnable state (Fig. 12.4). If we attempt to use any other method at this stage, an exception will be thrown.

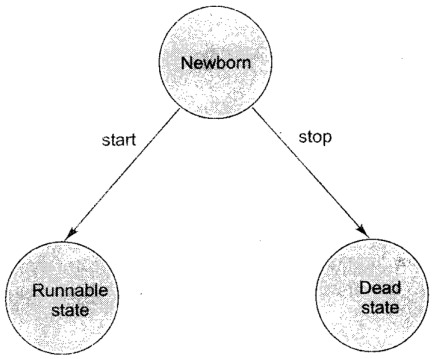



**Fig. 12.4** *Scheduling a newborn thread*

### Runnable State

The *runnable* state means that the thread is ready for execution and is waiting for the availability of the processor. That is, the thread has joined the queue of threads that are waiting for execution. If all threads have equal priority, then they are given time slots for execution in round robin fashion, i.e., first-come, first-serve manner. The thread that relinquishes control joins the queue at the end and again waits for its turn. This process of assigning time to threads is known as *time-slicing*.

However, if we want a thread to relinquish control to another thread to equal priority before its turn comes, we can do so by using the **yield( )** method (Fig. 12.5).

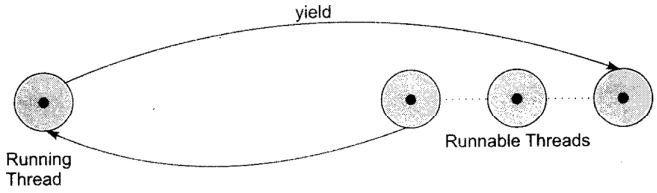



**Fig. 12.5** *Relinquishing control using yield( ) method*

### Running State

*Running* means that the processor has given its time to the thread for its execution. The thread runs until it relinquishes control on its own or it is preempted by a higher priority thread. A running thread may relinquish its control in one of the following situations.

1. It has been suspended using **suspend( )** method. A suspended thread can be revived by using the **resume( )** method. This approach is useful when we want to suspend a thread for some time due to certain reason, but do not want to kill it.

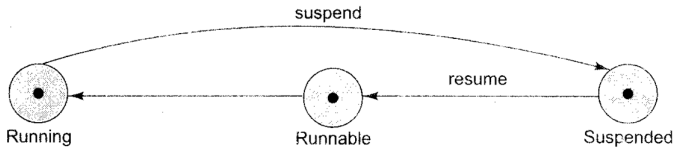



**Fig. 12.6** *Relinquishing control using suspend( ) method*

2. It has been made to sleep. We can put a thread to sleep for a specified time period using the method **sleep**(*time*) where *time* is in milliseconds. This means that the thread is out of the queue during this time period. The thread re-enters the runnable state as soon as this time period is elapsed.

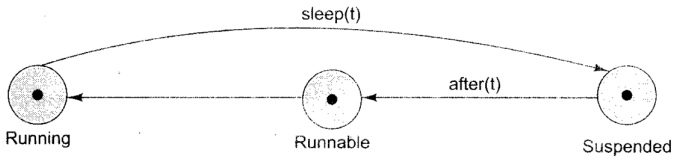



**Fig. 12.7** *Relinquishing control using sleep( ) method*

3. It has been told to wait until some event occurs. This is done using the **wait( )** method. The thread can be scheduled to run again using the **notify( )** method.

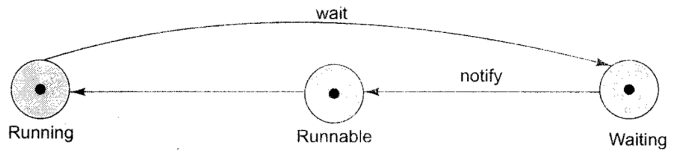



**Fig. 12.8** *Relinquishing control using wait( ) method*

### Blocked State

A thread is said to be *blocked* when it is prevented from entering into the runnable state and subsequently the running state. This happens when the thread is suspended, sleeping, or waiting in order to satisfy certain requirements. A blocked thread is considered "not runnable" but not dead and therefore fully qualified to run again.

### Dead State

Every thread has a life cycle. A running thread ends its life when it has completed executing its **run( )** method. It is a natural death. However, we can kill it by sending the stop message to it at any state thus causing a premature death to it. A thread can be killed as soon it is born, or while it is running, or even when it is in "not runnable" (blocked) condition.

## 12.6 Using Thread Methods

We have discussed how **Thread** class methods can be used to control the behaviour of a thread. We have used the methods **start( )** and **run( )** in Program 12.1. There are also methods that can move a thread from one state to another. Program 12.2 illustrates the use of **yield( ), sleep( )** and **stop( )** methods. Compare the outputs of Programs 12.1 and 12.2.

**Program 12.2** ***Use of yield( ), stop( ), and sleep( ) methods***

```
class A extends Thread
{
    public void run( )
    {
        for(int i = 1; i<=5; i++)
        {
            if(i==1) yield( );
            System.out.println("\tFrom Thread A : i = " +i);
        }
      System.out.println("exit from A ");
    }
}
class B extends Thread
{
    public void run( )
        {
            for(int i=1; i<=5; j++)
            {
                System.out.println("\tFrom Thread B : j = " + j);
                if(j==3) stop( );
        }
```

*(Continued)*

**Program 12.2** *(Continued)*

```
            System.out.println("Exit from B ");
        }
    }
    class C extends Thread
    {
        public void run( )
        {
            for (int k=1; k<=5; k++)
            {
                System.out.println("\tFrom Thread C : k = " +k);
                if(k==1)
                try
                    {
                        sleep(1000);
                    }
                catch (Exception e)
                {
                }
            }
            System.out.println("Exit from C ");
        }
    }
    class ThreadMethods
    {
        public static void main(String args[ ])
        {
            A threadA = new A( );
            B threadB = new B( );
            C threadC = new C( );

            System.out.println("Start thread A");
            threadA.start( );

            System.out.println("Start thread B");
            threadB.start( );

            System.out.println("Start thread C");
            threadC.start( );

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

Here is the output of Program 12.2:

```
Start thread A
Start thread B
Start thread C
    From Thread B : j = 1
    From Thread B : j = 2
    From Thread A : i = 1
    From Thread A : i = 2
End of main thread
    From Thread C : k = 1
    From Thread B : j = 3
    From Thread A : i = 3
    From Thread A : i = 4
    From Thread A : i = 5
Exit from A
    From Thread C : k = 2
    From Thread C : k = 3
    From Thread C : k = 4
    From Thread C : k = 5
Exit from C
```

Program 12.2 uses the **yield( )** method in thread A at the iteration i = 1. Therefore, the thread A, although started first, has relinquished its control to the thread B. The **stop( )** method in thread B has killed it after implementing the **for** loop only three times. Note that it has not reached end of **run( )** method. The thread C started sleeping after executing the for loop only once. When it woke up (after 1000 milliseconds), the other two threads have already completed their runs and therefore was running alone. The main thread died much earlier than the other three threads.

## 12.7 Thread Exceptions

Note that the call to **sleep( )** method is enclosed in a **try** block and followed by a **catch** block. This is necessary because the **sleep( )** method throws an exception, which should be caught. If we fail to catch the exception, program will not compile.

Java run system will throw **IllegalThreadStateException** whenever we attempt to invoke a method that a thread cannot handle in the given state. For example, a sleeping thread cannot deal with the **resume( )** method because a sleeping thread cannot receive any instructions. The same is true with the **suspend( )** method when it is used on a blocked (Not Runnable) thread.

Whenever we call a thread method that is likely to throw an exception, we have to supply an appropriate exception handler to catch it. The **catch** statement may take one of the following forms:

```
catch (ThreadDeath e)
{
    ..........
    ..........          // Killed thread
}
```

```
catch (InterruptedException e)
{
    ..........
    ..........          // Cannot handle it in the current state
}
catch (IllegalArgumentException e)
{
    ..........
    ..........          // Illegal method argument
}
catch (Exception e)
{
    ..........
    ..........          // Any other
}
```

Exception handling is discussed in detail in Chapter 13.

## 12.8 Thread Priority

In Java, each thread is assigned a priority, which affects the order in which it is scheduled for running. The threads that we have discussed so far are of the same priority. The threads of the same priority are given equal treatment by the Java scheduler and, therefore, they share the processor on a first-come, first-serve basis.

Java permits us to set the priority of a thread using the **setPriority( )** method as follows:

```
ThreadName.setPriority(intNumber);
```

The **intNumber** is an integer value to which the thread's priority is set. The **Thread** class defines several priority constants:

```
MIN_PRIORITY    =        1
NORM_PRIORITY   =        5
MAX_PRIORITY    =        10
```

The **intNumber** may assume one of these constants or any value between 1 and 10. Note that the default setting is NORM_PRIORITY.

Most user-level processes should use NORM_PRIORITY, plus or minus 1. Back-ground tasks such as network I/O and screen repainting should use a value very near to the lower limit. We should be very cautious when trying to use very high priority values. This may defeat the very purpose of using multithreads.

By assigning priorities to threads, we can ensure that they are given the attention (or lack of it) they deserve. For example, we may need to answer an input as quickly as possible. Whenever multiple threads are ready for execution, the Java system chooses the highest priority thread and executes it. For a thread of lower priority to gain control, one of the following things should happen:

1. It stops running at the end of **run( ).**
2. It is made to sleep using **sleep( ).**
3. It is told to wait using **wait( ).**

However, if another thread of a higher priority comes along, the currently running thread will be *preempted* by the incoming thread thus forcing the current thread to move to the runnable state. Remember that the highest priority thread always preempts any lower priority threads.

Program 12.3 and its output illustrate the effect of assigning higher priority to a thread. Note that although the thread A started first, the higher priority thread B has preempted it and started printing the output first. Immediately, the thread C that has been assigned the highest priority takes control over the other two threads. The thread A is the last to complete.

**Program 12.3** ***Use of priority in threads***

```
class A extends Thread
{
    public void run( )
    {
        System.out.println("threadA started");
        for(int i=1; i<=4; i++)
        {
            System.out.println("\tFrom Thread A : i = " +i);
        }
        System.out.println("Exit from A ");
    }
}
class B extends Thread
{
    public void run( )
    {
        System.out.println("threadB started");
        for(int j=1; j<=4; j++)
        {
            System.out.println("\tFrom Thread B : j = " + j);
        }
        System.out.println("Exit from B ");
    }
}
class C extends Thread
{
    public void run( )
    {
        System.out.println("threadC started");
        for(int k=1; k<=4; k++)
        {
            System.out.println("\tFrom Thread C : k = " + k);
    }
```

*(Continued)*

**Program 12.3** *(Continued)*

```
        System.out.println("Exit from C ");
        }
}
class ThreadPriority
{
    public static void main(String args[ ])
    {
        A threadA = new A( );
        B threadB = new B( );
        C threadC = new C( );

        threadC.setPriority(Thread.MAX_PRIORITY);
        threadB.setPriority(threadA.getPriority( )+1);
        threadA.setPrioirty(Thread.MIN_PRIORITY);

        System.out.println("Start thread A");
        threadA.start( );

        System.out.println("Start thread B");
        threadB.start( );

        System.out.println("Start thread C");
        threadC.start( );

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

Output of Program 12.3:

```
Start thread A
Start thread B
Start thread C
threadB started
    From Thread B : j = 1
    From Thread B : j = 2
threadC started
    From Thread C : k = 1
    From Thread C : k = 2
    From Thread C : k = 3
    From Thread C : k = 4
Exit from C
End of main thread
    From Thread B : j = 3
    From Thread B : j = 4
```

```
Exit from B
threadA started
     From Thread A : i = 1
     From Thread A : i = 2
     From Thread A : i = 3
     From Thread A : i = 4
Exit from A
```

**Threading using Interface**

Say our class is already extending another class. Now to extend this class as a child of Thread class is not possible, as Java does not support Multiple Inheritance. Hence Threading for classes that already extend a class has to be done differently. Lets see an example of the same

```
class A
{
   public void putdata(int i)
   {
      System.out.println("Printing "+i);
   }
}

class B extends A implements Runnable
{
   public B()
   {
      Thread t=new Thread(this);
      t.start();
   }

   public void run()
   {
      for(int i=1;i<=1000;i++)
         putdata(i);
   }
}

public class ex3
{
   public static void main(String args[ ])
   {
      B b1=new B();
      B b2=new B();
   }
}
```

Here we make use of an interface called Runnable and we create a Thread object 't' that is used to convert the object ('this' i.e. calling object) into an instance of thread & then the thread is started.

# Chapter 11

# Applets

## 14.2 How Applets Differ from Applications

Although both the applets and stand-alone applications are Java programs, there are significant differences between them. Applets are not full-featured application programs. They are usually written to accomplish a small task or a component of a task. Since they are usually designed for use on the Internet, they impose certain limitations and restrictions in their design.

- Applets 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 stand-alone applications, applets cannot be run independently. They are run from inside a Web page using a special feature known as HTML tag.
- Applets cannot read from or write to the files in the local computer.
- Applets cannot communicate with other servers on the network.
- Applets cannot run any program from the local computer.
- Applets are restricted from using libraries from other languages such as C or C++. (Remember, Java language supports this feature through **native** methods).

All these restrictions and limitations are placed in the interest of security of systems. These restrictions ensure that an applet cannot do any damage to the local system.

## 14.5 Applet Life Cycle

Every Java applet inherits a set of default behaviours from the **Applet** class. As a result, when an applet is loaded, it undergoes a series of changes in its state as shown in Fig. 14.5. The applet states include:

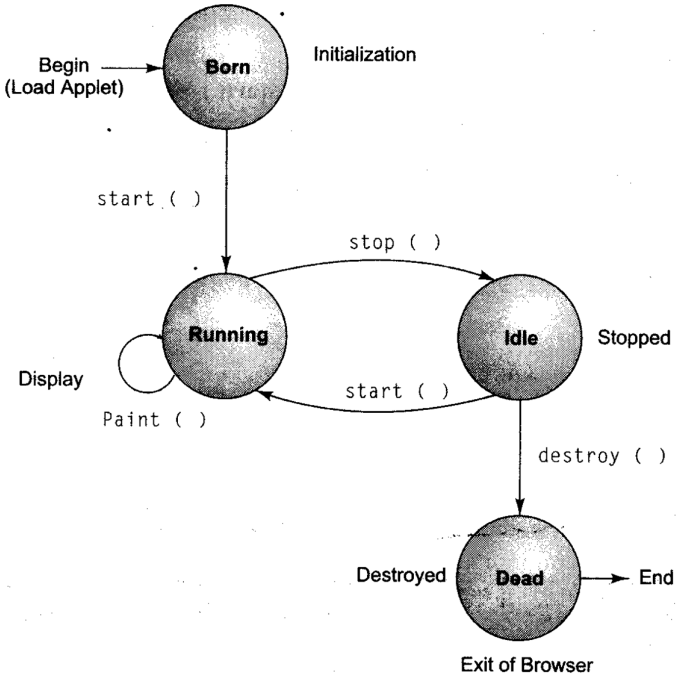



**Fig. 14.5** *An applet's state transition diagram*

- Born on initialization state
- Running state
- Idle state
- Dead or destroyed state

### Initialization State

Applet enters the ***initialization*** state when it is first loaded. This is achieved by calling the **init( )** method of **Applet** Class. The applet is born. At this stage, we may do the following, if required.

- Create objects needed by the applet
- Set up initial values
- Load images or fonts
- Set up colors

The initialization occurs only once in the applet's life cycle. To provide any of the behaviours mentioned above, we must override the **init( )** method:

```
public void init( )
{
    ..........
    .......... (Action)
    ..........
}
```

### Running State

Applet enters the ***running*** state when the system calls the **start( )** method of **Applet** Class. This occurs automatically after the applet is initialized. Starting can also occur if the applet is already in 'stopped" (idle) state. For example, we may leave the Web page containing the applet temporarily to another page and return back to the page. This again starts the applet running. Note that, unlike **init( )** method, the **start( )** method may be called more than once. We may override the **start( )** method to create a thread to control the applet.

```
public voids start( )
{
    ..........
    .......... (Action)
    ..........
}
```

### Idle or Stopped State

An applet becomes *idle* when it is stopped from running. Stopping occurs automatically when we leave the page containing the currently running applet. We can also do so by calling the **stop( )** method explicitly. If we use a thread to run the applet, then we must use **stop( )** method to terminate the thread. We can achieve this by overriding the **stop( )** method;

```
public void stop( )
{
    ..........
    .......... (Action)
    ..........
}
```

### Dead State

An applet is said to be *dead* when it is removed from memory. This occurs automatically by invoking the **destroy( )** method when we quit the browser. Like initialization, destroying stage occurs only once in the applet's life cycle. If the applet has created any resources, like threads, we may override the **destroy( )** method to clean up these resources.

```
public void destroy( )
{
    ..........
    .......... (Action)
    ..........
}
```

### Display State

Applet moves to the *display* state whenever it has to perform some output operations on the screen. This happens immediately after the applet enters into the running state. The **paint( )** method is called to accomplish this task. Almost every applet will have a **paint( )** method. Like other methods in the life cycle, the default version of **paint( )** method does absolutely nothing. We must therefore override this method if we want anything to be displayed on the screen.

```
public void paint (Graphics g)
{
    .....
    ..... (Display statements)
    .....
}
```

It is to be noted that the display state is not considered as a part of the applet's life cycle. In fact, the **paint( )** method is defined in the **Applet** class. It is inherited from the **Component** class, a super class of **Applet.**

## 14.12 Passing Parameters to Applets

We can supply user-defined parameters to an applet using <PARAM...> tags. Each <PARAM...> tag has a ***name*** attribute such as ***color***, and a ***value*** attribute such as ***red***. Inside the applet code, the applet can refer to that parameter by name to find its value. For example, we can change the colour of the text displayed to red by an applet by using a <PARAM...> tag as follows:

```
<APPLET .....>
<PARAM = color VALUE = "red">
</APPLET>
```

Similarly, we can change the text to be displayed by an applet by supplying new text to the applet through a <PARAM...> tag as shown below:

```
<PARAM NAME = text VALUE = "I love Java">
```

Passing parameters to an applet code using <PARAM> tag is something similar to passing parameters to the **main( )** method using command line arguments. To set up and handle parameters, we need to do two things:

1. Include appropriate <PARAM...> tags in the HTML document.
2. Provide Code in the applet to parse these parameters.

Parameters are passed on an applet when it is loaded. We can define the **init( )** method in the applet to get hold of the parameters defined in the <PARAM> tags. This is done using the **getParameter( )** method, which takes one string argument representing the **name** of the parameter and returns a string containing the value of that parameter.

Program 14.2 shows another version of **HelloJava** applet. Compile it so that we have a class file ready.

**Program 14.2** ***Applet HelloJavaParam***

```
import java.awt.*;
import java.applet.*;
public class HelloJavaParam extends Applet
{
  String str;
  public void init( )
  {
      str = getParameter("string");    // Receiving parameter value
      if (str == null)
          str = "Java";
      str = "Hello" + str;             // Using the value
  }
  public void paint (Graphics g)
  {
      g.drawString(str, 10, 100).
  }
}
```

Now, let us create HTML file that contains this applet. Program 14.3 shows a Web page that passes a parameter whose NAME is "string" and whose VALUE is "APPLET!" to the applet **HelloJavaParam**.

**Program 14.3** ***The HTML file for HelloJavaParam applet***

```
<HTML>
      <! Parameterized HTML file >
      <HEAD>
      <TITLE> Welcome to Java Applets </TITLE>
      <HEAD>
      <BODY>
            <APPLET CODE = HelloJavaParam.class
                  WIDTH = 400
                  HEIGHT = 200>
            <PARAM NAME = "string"
                  VALUE = "Applet! ">
            </APPLET>
      </BODY>
</HTML>
```

Save this file as **HelloJavaParam.html** and then run the applet using the applet viewer as follows:

```
appletviewer HelloJavaParam.html
```

This will produce the result as shown in Fig. 14.8.

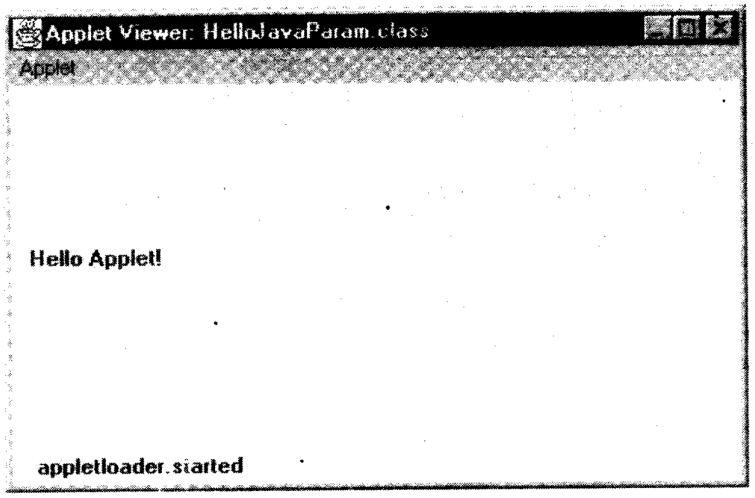



**Fig. 14.8** *Displays of HelloJavaParam applet*

Now, remove the <PARAM> tag from the HTML file and then run the applet again. The result wil be as shown in Fig. 14.9.

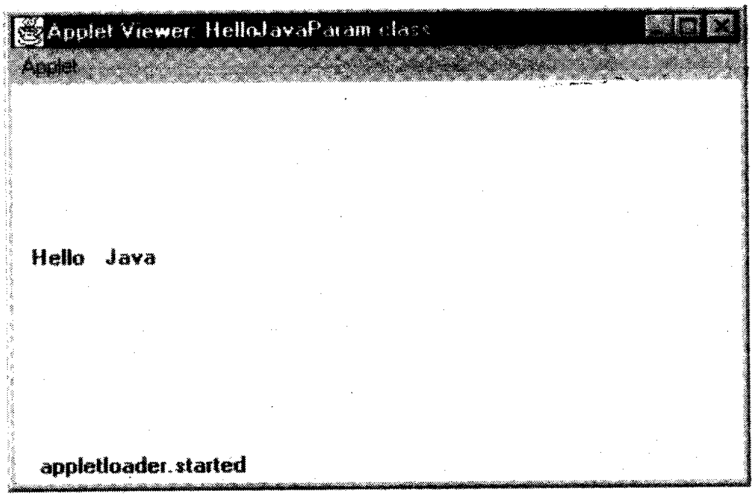



**Fig. 14.9** *Output without PARAM tag*

# COLOR EXAMPLE

```
import java.awt.*;
import java.applet.*;
public class showColor extends Applet
{
 //init set window dimensions
 public void init()
 {
        setSize( 400, 130 );
 }

 // draw rectangles and Strings in different colors
 public void paint( Graphics g )
 {
        // set new drawing color using integers
        g.setColor( new Color( 255, 0, 0 ) );
        g.fillRect( 25, 25, 100, 20 );
        g.drawString( "Current RGB: " + g.getColor(), 130, 40 );

        // set new drawing color using floats
        g.setColor( new Color( 0.0f, 1.0f, 0.0f ) );
        g.fillRect( 25, 50, 100, 20 );
        g.drawString( "Current RGB: " + g.getColor(), 130, 65 );

        // set new drawing color using static Color objects
        g.setColor( Color.blue );
        g.fillRect( 25, 75, 100, 20 );
        g.drawString( "Current RGB: " + g.getColor(), 130, 90 );

        // display individual RGB values
        Color color = Color.magenta;
        g.setColor( color );
        g.fillRect( 25, 100, 100, 20 );
        g.drawString( "RGB values: " + color.getRed() + ", " + color.getGreen() + ",
" + color.getBlue(), 130, 115 );
 }
 } // end class ShowColors
```

| Color Constant | Color | RGB value |
|---|---|---|
| `public final static Color orange` | orange | 255, 200, 0 |
| `public final static Color pink` | pink | 255, 175, 175 |
| `public final static Color cyan` | cyan | 0, 255, 255 |
| `public final static Color magenta` | magenta | 255, 0, 255 |
| `public final static Color yellow` | yellow | 255, 255, 0 |
| `public final static Color black` | black | 0, 0, 0 |
| `public final static Color white` | white | 255, 255, 255 |
| `public final static Color gray` | gray | 128, 128, 128 |
| `public final static Color lightGray` | light gray | 192, 192, 192 |
| `public final static Color darkGray` | dark gray | 64, 64, 64 |
| `public final static Color red` | red | 255, 0, 0 |
| `public final static Color green` | green | 0, 255, 0 |
| `public final static Color blue` | blue | 0, 0, 255 |

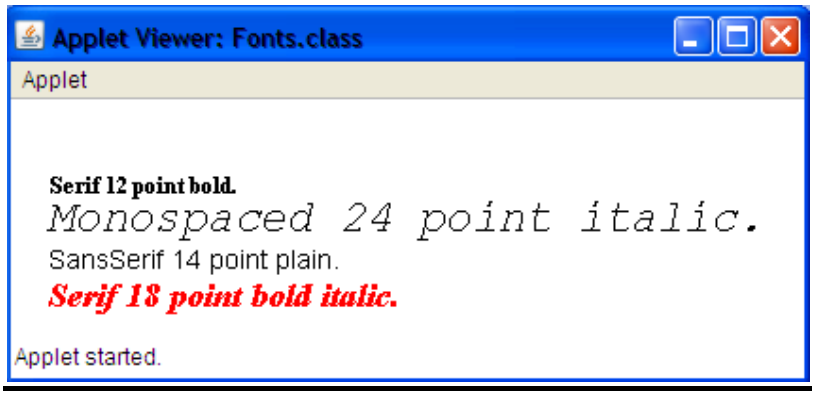
Fig. 11.3 `Color` class `static` constants and RGB values

| Method | Description |
|---|---|
| `public Color( int r, int g, int b )` | Creates a color based on red, green and blue contents expressed as integers from 0 to 255. |
| `public Color( float r, float g, float b )` | Creates a color based on red, green and blue contents expressed as floating-point values from 0.0 to 1.0. |
| `public int getRed()  // Color class` | Returns a value between 0 and 255 representing the red content. |
| `public int getGreen()  // Color class` | Returns a value between 0 and 255 representing the green content. |
| `public int getBlue()  // Color class` | Returns a value between 0 and 255 representing the blue content. |
| `public Color getColor()  // Graphics class` | Returns a `Color` object representing the current color for the graphics context. |
| `public void setColor( Color c ) // Graphics class` | Sets the current color for drawing with the graphics context. |

Fig. 11.4 `Color` methods and color-related `Graphics` methods .

# FONT EXAMPLE

```
import java.awt.*;
import java.applet.*;

public class Fonts extends Applet
{
        public void init()
        {
                setSize( 420, 125 );
        }

        // display Strings in different fonts and colors
        public void paint( Graphics g )
        {
                // set current font to Serif (Times), bold, 12pt
                // and draw a string
                g.setFont( new Font( "Serif", Font.BOLD, 12 ) );
                g.drawString( "Serif 12 point bold.", 20, 50 );

                // set current font to Monospaced (Courier),
                // italic, 24pt and draw a string
                g.setFont(new Font( "Monospaced", Font.ITALIC, 24 ));
                g.drawString( "Monospaced 24 point italic.", 20, 70 );

                // set current font to SansSerif (Helvetica),
                // plain, 14pt and draw a string
                g.setFont( new Font( "SansSerif", Font.PLAIN, 14 ) );
                g.drawString( "SansSerif 14 point plain.", 20, 90 );

                // set current font to Serif (times), bold/italic,
                // 18pt and draw a string
                g.setColor( Color.red );
            g.setFont(new Font( "Serif",Font.BOLD+Font.ITALIC,18));
                g.drawString( g.getFont().getName() + " " +
                g.getFont().getSize() + " point bold italic.", 20, 110 );
        }
 } // end class Fonts
```




| Method or constant | Description |
|---|---|
| `public final static int PLAIN // Font class` | A constant representing a plain font style. |
| `public final static int BOLD // Font class` | A constant representing a bold font style. |
| `public final static int ITALIC // Font class` | A constant representing an italic font style. |
| `public Font( String name, int style, int size )` | Creates a **Font** object with the specified font, style and size. |
| `public int getStyle() // Font class` | Returns an integer value indicating the current font style. |
| `public int getSize() // Font class` | Returns an integer value indicating the current font size. |
| `public String getName() // Font class` | Returns the current font name as a string. |
| `public String getFamily() // Font class` | Returns the font's family name as a string. |
| `public boolean isPlain() // Font class` | Tests a font for a plain font style. Returns **true** if the font is plain. |
| `public boolean isBold() // Font class` | Tests a font for a bold font style. Returns **true** if the font is bold. |
| `public boolean isItalic() // Font class` | Tests a font for an italic font style. Returns **true** if the font is italic. |
| `public Font getFont() // Graphics class` | Returns a **Font** object reference representing the current font. |
| `public void setFont( Font f ) // Graphics class` | Sets the current font to the font, style and size specified by the **Font** object reference **f**. |

Fig. 11.8 **Font** methods, constants and font-related **Graphics** methods .

| Method | Description |
|---|---|
| `public void drawLine( int x1, int y1, int x2, int y2 )` | Draws a line between the point (**x1**, **y1**) and the point (**x2**, **y2**). |
| `public void drawRect( int x, int y, int width, int height )` | Draws a rectangle of the specified **width** and **height**. The top-left corner of the rectangle has the coordinates (**x**, **y**). |
| `public void fillRect( int x, int y, int width, int height )` | Draws a solid rectangle with the specified **width** and **height**. The top-left corner of the rectangle has the coordinate (**x**, **y**). |
| `public void clearRect( int x, int y, int width, int height )` | Draws a solid rectangle with the specified **width** and **height** in the current background color. The top-left corner of the rectangle has the coordinate (**x**, **y**). |
| `public void drawRoundRect( int x, int y, int width, int height, int arcWidth, int arcHeight )` | Draws a rectangle with rounded corners in the current color with the specified **width** and **height**. The **arcWidth** and **arcHeight** determine the rounding of the corners (see Fig. 11.15). |
| `public void fillRoundRect( int x, int y, int width, int height, int arcWidth, int arcHeight )` | Draws a solid rectangle with rounded corners in the current color with the specified **width** and **height**. The **arcWidth** and **arcHeight** determine the rounding of the corners (see Fig. 11.15). |

`public void draw3DRect( int x, int y, int width, int height, boolean b )`

Draws a three-dimensional rectangle in the current color with the specified **width** and **height**. The top-left corner of the rectangle has the coordinates (**x**, **y**). The rectangle appears raised when b is true and is lowered when b is false.

`public void fill3DRect( int x, int y, int width, int height, boolean b )`

Draws a filled three-dimensional rectangle in the current color with the specified **width** and **height**. The top-left corner of the rectangle has the coordinates (**x**, **y**). The rectangle appears raised when b is true and is lowered when b is false.

`public void drawOval( int x, int y, int width, int height )`

Draws an oval in the current color with the specified **width** and **height**. The bounding rectangle's top-left corner is at the coordinates (**x**, **y**). The oval touches all four sides of the bounding rectangle at the center of each side (see Fig. 11.16).

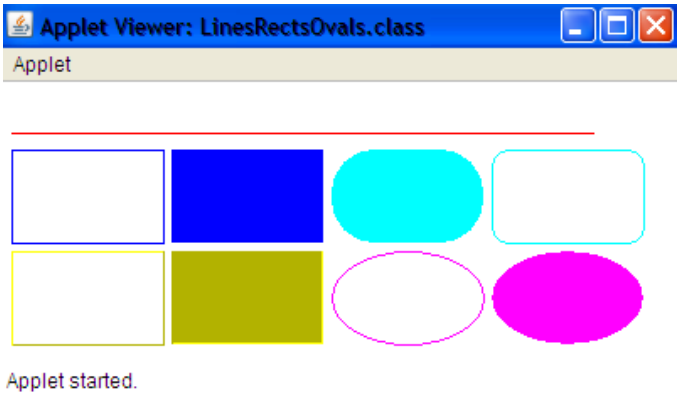
`public void fillOval( int x, int y, int width, int height )`

Draws a filled oval in the current color with the specified **width** and **height**. The bounding rectangle's top-left corner is at the coordinates (**x**, **y**). The oval touches all four sides of the bounding rectangle at the center of each side (see Fig. 11.16).

Fig. 11.13 **Graphics** methods that draw lines, rectangles and ovals (part 2 of 2).

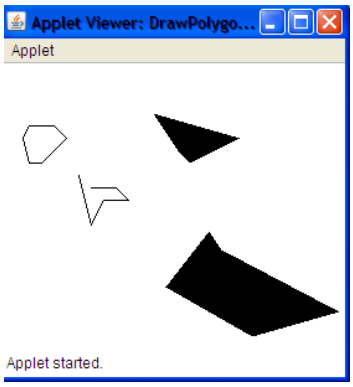
# LINES – RECTANGLES - OVALS

```
import java.awt.*;
import java.applet.*;
 public class LinesRectsOvals extends Applet
 {
        public void init()
       {
              setSize( 400, 165 );
       }
        // display various lines, rectangles and ovals
        public void paint( Graphics g )
        {
               g.setColor( Color.red );
               g.drawLine( 5, 30, 350, 30 );
               g.setColor( Color.blue );
               g.drawRect( 5, 40, 90, 55 );
               g.fillRect( 100, 40, 90, 55 );

               g.setColor( Color.cyan );
               g.fillRoundRect( 195, 40, 90, 55, 50, 50 );
               g.drawRoundRect( 290, 40, 90, 55, 20, 20 );

               g.setColor( Color.yellow );
               g.draw3DRect( 5, 100, 90, 55, true );
               g.fill3DRect( 100, 100, 90, 55, false );

               g.setColor( Color.magenta );
               g.drawOval( 195, 100, 90, 55 );
               g.fillOval( 290, 100, 90, 55 );
        }
}
```



# POLYGONS

```
import java.awt.*;
import java.applet.*;

public class DrawPolygons extends Applet
{
        // set window's title bar String and dimensions
        public void init()
        {
                setSize( 275, 230 );
        }

        // draw polygons and polylines
        public void paint( Graphics g )
        {
                int xValues[] = { 20, 40, 50, 30, 20, 15 };
                int yValues[] = { 50, 50, 60, 80, 80, 60 };
                Polygon polygon1 = new Polygon( xValues, yValues, 6 );

                g.drawPolygon( polygon1 );
                int xValues2[] = { 70, 90, 100, 80, 70, 65, 60 };
                int yValues2[] = { 100, 100, 110, 110, 130, 110, 90 };
                g.drawPolyline( xValues2, yValues2, 7 );

                int xValues3[] = { 120, 140, 150, 190 };
                int yValues3[] = { 40, 70, 80, 60 };
                g.fillPolygon( xValues3, yValues3, 4 );

                Polygon polygon2 = new Polygon();
                polygon2.addPoint( 165, 135 );
                polygon2.addPoint( 175, 150 );
                polygon2.addPoint( 270, 200 );
                polygon2.addPoint( 200, 220 );
                polygon2.addPoint( 130, 180 );
                g.fillPolygon( polygon2 );
        }
}
```



| Method | Description |
|---|---|

```
public void drawPolygon( int xPoints[], int yPoints[],
   int points )
```

Draws a polygon. The *x*-coordinate of each point is specified in the **xPoints** array and the *y*-coordinate of each point is specified in the **yPoints** array. The last argument specifies the number of **points**. This method draws a closed polygon—even if the last point is different from the first point.

```
public void drawPolyline( int xPoints[], int yPoints[],
   int points )
```

Draws a series of connected lines. The *x*-coordinate of each point is specified in the **xPoints** array and the *y*-coordinate of each point is specified in the **yPoints** array. The last argument specifies the number of **points**. If the last point is different from the first point, the polyline is not closed.

```
public void drawPolygon( Polygon p )
```

Draws the specified closed polygon.

```
public void fillPolygon( int xPoints[], int yPoints[],
   int points )
```

Draws a solid polygon. The *x*-coordinate of each point is specified in the **xPoints** array and the *y*-coordinate of each point is specified in the **yPoints** array. The last argument specifies the number of **points**. This method draws a closed polygon—even if the last point is different from the first point.

```
public void fillPolygon( Polygon p )
```

Draws the specified solid polygon. The polygon is closed.

```
public Polygon()                                    // Polygon class
```

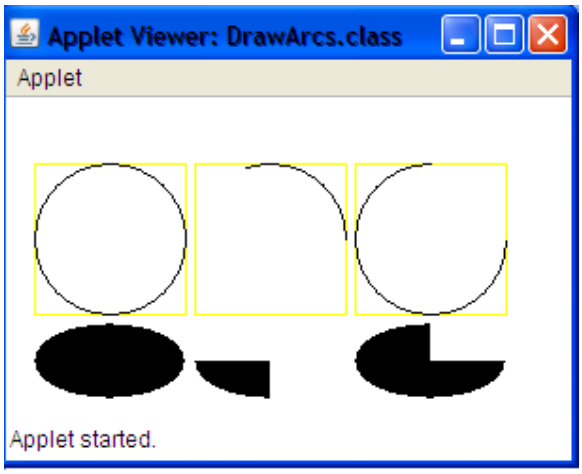
Constructs a new polygon object. The polygon does not contain any points.

```
public Polygon( int xValues[], int yValues[],       // Polygon class
   int numberOfPoints )
```

Constructs a new polygon object. The polygon has **numberOfPoints** sides, with each point consisting of an *x*-coordinate from **xValues** and a *y*-coordinate from **yValues**.

# DRAW ARCS

```
import java.awt.*;
import java.applet.*;
public class DrawArcs extends Applet
{
        public void init()
        {
                setSize( 300, 170 );
        }

         // draw rectangles and arcs
         public void paint( Graphics g )
         {
                 // start at 0 and sweep 360 degrees
                 g.setColor( Color.yellow );
                 g.drawRect( 15, 35, 80, 80 );
                 g.setColor( Color.black );
                 g.drawArc( 15, 35, 80, 80, 0, 360 );

                 // start at 0 and sweep 110 degrees
                 g.setColor( Color.yellow );
                 g.drawRect( 100, 35, 80, 80 );
                 g.setColor( Color.black );
                 g.drawArc( 100, 35, 80, 80, 0, 110 );
                 // start at 0 and sweep -270 degrees
                         g.setColor( Color.yellow );
                         g.drawRect( 185, 35, 80, 80 );
                         g.setColor( Color.black );
                         g.drawArc( 185, 35, 80, 80, 0, -270 );
                 // start at 0 and sweep 360 degrees
                         g.fillArc( 15, 120, 80, 40, 0, 360 );
                 // start at 270 and sweep -90 degrees
                         g.fillArc( 100, 120, 80, 40, 270, -90 );
                 // start at 0 and sweep -270 degrees
                         g.fillArc( 185, 120, 80, 40, 0, -270 );
        }
}
```



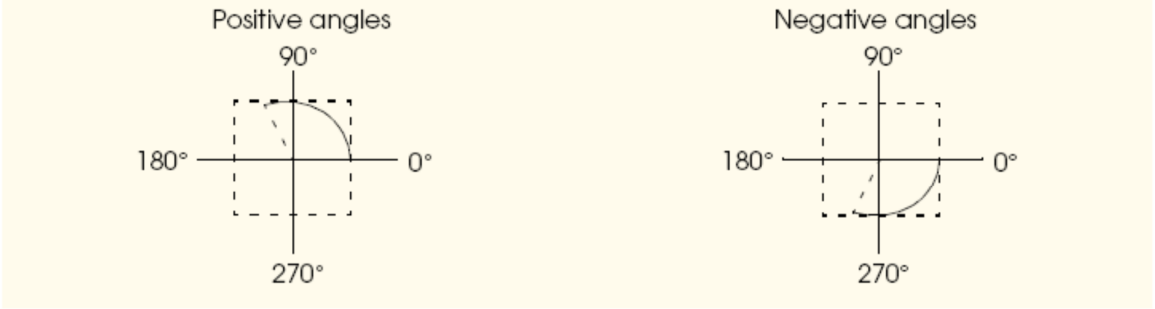




**Fig. 11.17** Positive and negative arc angles.

| Method | Description |
|---|---|

```
public void drawArc( int x, int y, int width, int height,
   int startAngle, int arcAngle )
```

Draws an arc relative to the bounding rectangle's top-left coordinates *(**x**, **y**)* with the specified **width** and **height**. The arc segment is drawn starting at startAngle and sweeps arcAngle degrees.

```
public void fillArc( int x, int y, int width, int height,
   int startAngle, int arcAngle )
```

Draws a solid arc (i.e., a sector) relative to the bounding rectangle's top-left coordinates *(**x**, **y**)* with the specified **width** and **height**. The arc segment is drawn starting at startAngle and sweeps arcAngle degrees.

**Fig. 11.18** **Graphics** methods for drawing arcs.

# IMAGES

```
import java.awt.*;
import java.applet.*;
public class Images extends Applet
{
   Image image;

      public void init()
    {  image = getImage(getCodeBase(), "p.JPG");
       setSize(400,500);
    }

       public void paint(Graphics g)
     {
        g.drawImage(image, 100, 10, this);
     }
}
```

# Chapter 12

# Theory

Type 1:

1. What is meant by method overriding?
2. What do you mean by method overloading?
3. Compare and contrast overloading and overriding methods.
4. What are abstract classes? When do we declare a method or class final/abstract?
5. Write a note on java access specifiers.

Type 2:

1. What is a Java interface? Explain with an example. What are the similarities and differences between interfaces and classes?
2. Write a Java program to illustrate the concept of multiple interface implementation. Also illustrate how an interface can be extended

Type 3:

1. What is meant by exception handling? –OR – What is an Exception? Explain Exception handling mechanism in Java
2. Write a Java program to illustrate the syntax of multiple catch statements after a try block.
3. What is thread? What is meant by multithreading?
4. Describe the complete life cycle of a thread.
5. What are the 2 methods of creating threads?
6. What is synchronization? When do we use it?

Type 4:

1. Explain applet life cycle with state transition diagram.
2. How is applet included in a HTML file?
3. Java applets generally do not have a main() method. What is the lifecycle of a Java applet and what standard methods do we have to implement? What order are they called in? If we do not implement one of these methods, will the Applet compile?
4. What do the init ( ) and start ( ) method do?
5. Write a difference between application program and Applet program. OR Differentiate: Java Applet and Java Application.

Type 5:

1. Explain any two java utility classes

(S1/T1/Q1). What is meant by method overridding?
Ans:
Anytime you have a class that inherits a method from a superclass, you have the opportunity to override the method (unless the method is marked final). The key benefit of overriding is the ability to define behavior that's specific to a particular subclass type.

The rules for overriding a method are as follows:
■ The argument list must exactly match that of the overridden method.
■ The return type must exactly match that of the overridden method.
■ The access level must not be more restrictive than that of the overridden method. (e.g. the original overridden function is public & the new subclass overriding method is private is not allowed).
■ The access level can be less restrictive than that of the overridden method(e.g. original overridden method is protected & the new subclass method overriding it is public is allowed).
■ The overriding method must not throw new or broader checked exceptions than those declared by the overridden method.
■ The overriding method can throw narrower or fewer exceptions. Bottom line: An overriding method doesn't have to declare any exceptions that it will never throw, regardless of what the overridden method declares.
■ You cannot override a method marked final.
■ If a method can't be inherited, you cannot override it.

Often, you'll want to take advantage of some of the code in the superclass version of a method, yet still override it to provide some additional specific behavior. It's like saying, "Run the superclass version of the method, then come back down here and finish with my subclass additional method code.". It's easy to do in code using the keyword 'super'.

Example: A program in JAVA to create a class employee which maintains the name, id & salary of the employee. Create another class manager which additionally maintains the designation of that manager. Accept details of two managers & print the details:

```
import java.io.*;
class Employee
{
   private String name;
   private int id;
   private double sal;
   public void getdata( ) throws IOException
   {
         BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
         System.out.println("Enter name, id, salary:");
         name=br.readLine();
         id=Integer.parseInt(br.readLine());
         sal=Double.parseDouble(br.readLine());
   }

  public void putdata( )
   {
      System.out.println("Name="+name);
      System.out.println("ID="+id);
      System.out.println("Salary="+sal);
   }
}
class Manager extends Employee
{
   private String desig;
   public void getdata() throws IOException //method or function overridding
   {
      super.getdata();
      BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Enter designation");
      desig=br.readLine();
   }
   public void putdata()
   {     super.putdata();
         System.out.println("Designation="+desig);
   }
}
public class ex
{
   public static void main(String args[ ]) throws IOException
   {
         Manager X=new Manager();
         Manager Y=new Manager();
         X.getdata();
         Y.getdata();
         X.putdata();
         Y.putdata();
   }
}
```

(S1/T1/Q2). What is meant by method overloading?
Ans:
Overloaded methods let you reuse the same method name in a class, but with different arguments.
The rules are simple:
- Overloaded methods must change the argument list.
- Overloaded methods can change the return type.
- Overloaded methods can change the access modifier.
- Overloaded methods can declare new or broader checked exceptions.
- A method can be overloaded in the same class or in a subclass.

Note: the only main criteria for method overloading is that they are multiple functions with the same name but something different in the parameter list – in number, in type, or in order of type (e.g. int,double & double,int).

When a method is invoked, more than one method of the same name might exist for the object type you're invoking a method on. Deciding which of the matching methods to invoke is based on the arguments. If you invoke the method with a String argument, the overloaded version that takes a String is called. If you invoke a method of the same name but pass it a *float*, the overloaded version that takes a *float* will run. If you invoke the method of that passes an object, then the overloaded method which takes the object of the corresponding type will be invoked. Antime if there isn't an overloaded version that takes that particular type of argument, then the compiler will complain that it can't find a match.

Example:

```
class Adder
{
        public int addThem(int x, int y)
        {
                return x + y;
        }
        // Overload the addThem method to add doubles instead of ints
        public double addThem(double x, double y)
        {
                return x + y;
        }

}
// From another class, invoke the addThem( ) method
public class TestAdder {
        public static void main (String [ ] args)
        {
                Adder a = new Adder();
                int b = 27;
                int c = 3;
                int result = a.addThem(b,c); // First addThem is invoked!
                double doubleResult = a.addThem(22.5,89.36);
                                        // Overloaded addThem is invoked!
```

```
        }
}
```

In the preceding TestAdder code, the first call to a.addThem(b,c) passes two *int*s to the method, so the first version of addThem()—the overloaded version that takes two *int* arguments—is called. The second call to a.addThem(22.5, 89.36) passes two *doubles* to the method, so the second version of addThem()—the overloaded version that takes two *double* arguments—is called.

NOTE: Function Overloading is COMPILE – TIME phenomenon & not a runtime phenomenon like function overridding. An example demonstrating this fact is given below:

```
class A
{
        public void display(int x)
        {
                System.out.println("A's x="+x);
        }
}

class B extends A
{
        public void display(double x)  //overloading
        {
                System.out.println("B's x="+x);
        }
}
```

```
public class set1_ex2
{
        public static void main(String args[ ])
        {
                B b=new B( );
                A a=b;
                a.display(2);
                b.display(2);
                a.display(2.2); //ERROR
                b.display(2.2); //calling the overloaded function
        }
}
```

**Note that function overloading is a compile time phenomenon hence a.display(2.2); will be class A's display (since only at runtime will it be known that 'a' is referencing an object of type B, but overloading is NOT a runtime phenomenon, hence it goes for class A and not B, since at compile time the compiler doesn't know that 'a' is referencing 'b'). Since class A does not have a function display that taken a double value hence a Compiler time ERROR.**

(S1/T1/Q3). Compare & Contrast Overloading & Overridding methods
Ans:

**TABLE 5-3** Difference Between Overloaded and Overridden Methods

| | Overloaded Method | Overridden Method |
|---|---|---|
| argument list | Must change | Must not change |
| return type | Can change | Must not change |
| exceptions | Can change | Can reduce or eliminate. Must not throw new or broader checked exceptions |
| access | Can change | Must not make more restrictive (can be less restrictive) |
| invocation | *Reference* type determines which overloaded version (based on declared argument types) is selected. Happens at *compile* time. The actual *method* that's invoked is still a virtual method invocation that happens at runtime, but the compiler will already know the *signature* of the method to be invoked. So at runtime, the argument match will already have been nailed down, just not the actual *class* in which the method lives. | *Object* type (in other words, *the type of the actual instance on the heap*) determines which method is selected. Happens at *runtime*. |

Explain more if the marks are more……..

(S1/T1/Q4). What are abstract classes? When do we declare a method or class abstract/final?
Ans:

- An abstract class can never be instantiated. Its sole purpose is to be extended (subclassed). If even a single method is abstract, the whole class must be declared abstract.

- An abstract method is a method that's been declared (as abstract) but not implemented. In other words, the method contains no functional code.

- You mark a method abstract when you want to force subclasses to provide the implementation.

- Any class that extends an abstract class must implement all abstract methods of the superclass. Unless the subclass is also abstract. The rule is The first concrete subclass of an abstract class must implement all abstract methods of the superclass.
  Concrete just means nonabstract, so if you have an abstract class extending another abstract class, the abstract subclass doesn't need to provide implementations for the inherited abstract methods. Sooner or later, though, somebody's going to make a nonabstract subclass (in other words, a class that can be instantiated), and that subclass will have to implement all the abstract methods from up the inheritance tree.

- A method can never, ever, ever be marked as both abstract and final, or both abstract and private.

- A final class is one which cannot be subclassed

- A final method is one which cannot be overridden (i.e. the subclass cannot redefine the function).

- You can't mark a class as both abstract and final. They have nearly opposite meanings. An abstract class must be subclassed, whereas a final class must not be subclassed. If you see this combination of abstract and final modifiers, used for a class or method declaration, the code will not compile.

Example of abstract:

```
abstract class Figure
{
        double dim1;
        double dim2;
        Figure(double a, double b)
        {
                dim1 = a;
                dim2 = b;
        }
        abstract double area( );        // area is now an abstract method
}
```

```
class Rectangle extends Figure
{
        Rectangle(double a, double b)
        {
                super(a, b);
        }
        // override area for rectangle
        double area( )
        {
                System.out.println("Inside Area for Rectangle.");
                return dim1 * dim2;
        }
}

class Triangle extends Figure
{
        Triangle(double a, double b)
        {
                super(a, b);
        }
        // override area for right triangle
        double area()
        {
                System.out.println("Inside Area for Triangle.");
                return dim1 * dim2 / 2;
        }
}


class ex
{
        public static void main(String args[ ])
        {
                // Figure f = new Figure(10, 10); // illegal
                Rectangle r = new Rectangle(9, 5);
                Triangle t = new Triangle(10, 8);
                Figure figref; // this is OK, no object is created
                figref = r;
                System.out.println("Area is " + figref.area( ));
                figref = t;
                System.out.println("Area is " + figref.area( ));
        }
}
```

Example of final method & class:

While method overriding is one of Java's most powerful features, there will be times when you will want to prevent it from occurring. To disallow a method from being overridden, specify final as a modifier at the start of its declaration. Methods declared as final cannot be overridden. The following fragment illustrates final:

```
class A
{
        final void meth( )
        {
        System.out.println("This is a final method.");
        }
}

class B extends A
{
        void meth( )
        {
                // ERROR! Can't override.
                System.out.println("Illegal!");
        }
}
```

Because meth( ) is declared as final, it cannot be overridden in B. If you attempt to do so, a compile-time error will result.

---

Sometimes you will want to prevent a class from being inherited. To do this, precede the class declaration with final. Declaring a class as final implicitly declares all of its methods as final, too.

```
final class A
{
        ……….
}
// The following class is illegal.
class B extends A  // ERROR! Can't subclass A
{
        ……..
}
```

As the comments imply, it is illegal for B to inherit A since A is declared as final.

(S1/T1/Q5). Write a note on Java Access Specifiers
Ans:
Access Protection
The class is Java's smallest unit of abstraction. Because of the interplay between classes and packages,
Java addresses four categories of visibility for class members:
• Subclasses in the same package
• Non-subclasses in the same package
• Subclasses in different packages
• Classes that are neither in the same package nor subclasses
The three access specifiers, private, public, and protected, provide a variety of ways to produce the many levels of access required by these categories.

| | Private | No modifier | Protected | Public |
|---|---|---|---|---|
| Same class | Yes | Yes | Yes | Yes |
| Same package subclass | No | Yes | Yes | Yes |
| Same package non-subclass | No | Yes | Yes | Yes |
| Different package subclass | No | No | Yes | Yes |
| Different package non-subclass | No | No | No | Yes |

While Java's access control mechanism may seem complicated, we can simplify it as follows. Anything declared public can be accessed from anywhere. Anything declared private cannot be seen outside of its class. When a member does not have an explicitaccess specification, it is visible to subclasses as well as to other classes in the same package. This is the default access. If you want to allow an element to be seen outside your current package, but only to classes that subclass your class directly, then declare
that element protected. A class has only two possible access levels: default and public. When a class is declared as public, it is accessible by any other code. If a class has default access, then it can only be accessed by other code within its same package.

Example:

This is file Protection.java:

```
package p1;
public class Protection
{
        int n = 1;
        private int n_pri = 2;
        protected int n_pro = 3;
        public int n_pub = 4;
        public Protection( )
        {
                System.out.println("base constructor");
                System.out.println("n = " + n);
                System.out.println("n_pri = " + n_pri);
                System.out.println("n_pro = " + n_pro);
                System.out.println("n_pub = " + n_pub);
        }
}
```

This is file Derived.java:

```
package p1;
class Derived extends Protection
{
        Derived( )
        {
                System.out.println("derived constructor");
                System.out.println("n = " + n);
                // class only
                // System.out.println("n_pri = " + n_pri);
                System.out.println("n_pro = " + n_pro);
                System.out.println("n_pub = " + n_pub);
        }
}
```

This is file SamePackage.java:

```
package p1;
class SamePackage
{
        SamePackage( )
        {
                Protection p = new Protection();
                System.out.println("same package constructor");
                System.out.println("n = " + p.n);
                // class only
                // System.out.println("n_pri = " + p.n_pri);
                System.out.println("n_pro = " + p.n_pro);
                System.out.println("n_pub = " + p.n_pub);
        }
}
```

This is file Protection2.java:

```
package p2;
class Protection2 extends p1.Protection
{
        Protection2( )
        {
                System.out.println("derived other package constructor");
                // class or package only
                // System.out.println("n = " + n);
                // class only
                // System.out.println("n_pri = " + n_pri);
                System.out.println("n_pro = " + n_pro);
                System.out.println("n_pub = " + n_pub);
        }
}
```

This is file OtherPackage.java:

```
package p2;
class OtherPackage
{
        OtherPackage( )
        {
                p1.Protection p = new p1.Protection();
                System.out.println("other package constructor");
                // class or package only
                // System.out.println("n = " + p.n);
                // class only
                // System.out.println("n_pri = " + p.n_pri);
                // class, subclass or package only
                // System.out.println("n_pro = " + p.n_pro);
                System.out.println("n_pub = " + p.n_pub);
                }
}
```

If you wish to try these two packages, here are two test files you can use. The one for package p1 is shown here:

```
package p1;
public class Demo
{
        public static void main(String args[ ])
        {
                Protection ob1 = new Protection();
                Derived ob2 = new Derived();
                SamePackage ob3 = new SamePackage();
        }
}
```

The test file for p2 is shown next:

```
package p2;
public class Demo
{
        public static void main(String args[ ])
        {
                Protection2 ob1 = new Protection2();
                OtherPackage ob2 = new OtherPackage();
        }
}
```

(S1/T2/Q1). What is a Java Interface? Explain with an example. What are the similarities and differences between interface and classes?
Ans:
Using interface, you can specify what a class must do, but not how it does it. Interfaces are syntactically similar to classes, but they lack instance variables, and their methods are declared without any body. In practice, this means that you can define interfaces which don't make assumptions about how they are implemented. Once it is defined, any number of classes can implement an interface. Also, one class can implement any number of interfaces.
To implement an interface, a class must create the complete set of methods defined by the interface. However, each class is free to determine the details of its own implementation.

Rules for interfaces:
■ All interface methods are implicitly public and abstract.
■ Interface methods must not be static.
■ You do not need to actually type the public or abstract modifiers in the method declaration, but the method is still always public and abstract.
■ All variables defined in an interface must be public, static, and final—in other words, interfaces can declare only constants, not instance variables.
■ An interface can extend one or more other interfaces.
■ An interface cannot extend anything but another interface.
■ An interface cannot implement another interface or class.
■ An interface must be declared with the keyword interface.

The general form of an interface:

```
access interface name
{
        return-type method-name1(parameter-list);
        return-type method-name2(parameter-list);
        type final-varname1 = value;
        type final-varname2 = value;
         .....
        return-type method-nameN(parameter-list);
        type final-varnameN = value;
}
```

Example:

```
interface Callback
{
        void callback(int param);
}

class Client implements Callback
{
        // Implement Callback's interface
        public void callback(int p)
        {
                System.out.println("callback called with " + p);
        }
}

// Another implementation of Callback.
class AnotherClient implements Callback
{
        // Implement Callback's interface
        public void callback(int p)
        {
                System.out.println("Another version of callback");
                System.out.println("p squared is " + (p*p));
        }
}

class ex
{
        public static void main(String args[ ])
        {
                Callback c = new Client();
                AnotherClient ob = new AnotherClient();
                c.callback(42);
                c = ob; // c now refers to AnotherClient object
                c.callback(42);
        }
}
```

The output from this program is shown here:

```
callback called with 42
Another version of callback
p squared is 1764
```

As you can see, the version of callback( ) that is called is determined by the type of object that c refers to at run time.

## Partial Implementations

If a class includes an interface but does not fully implement the methods defined by that interface, then that class must be declared as abstract.

For example:

```
abstract class Incomplete implements Callback
{
        int a, b;
        void show()
        {
                System.out.println(a + " " + b);
        }
        …….
}
```

Here, the class Incomplete does not implement callback( ) and must be declared as abstract. Any class that inherits Incomplete must implement callback( ) or be declared abstract itself.

## Variables in Interfaces

By placing the constants right in the interface, any class that implements the interfacehas direct access to the constants, just as if the class had inherited them.
They must always be

- public
- static
- final

Once the value has been assigned, the value can never be modified. The assignment happens in the interface itself (where the constant is declared), so the implementing class can access it and use it, but as a read-only value.

```
import java.util.Random;
interface SharedConstants
{
        int NO = 0;
        int YES = 1;
        int MAYBE = 2;
        int LATER = 3;
        int SOON = 4;
        int NEVER = 5;
}
class Question implements SharedConstants
{
        Random rand = new Random( );
        int ask( )
        {
                int prob = (int) (100 * rand.nextDouble());
                if (prob < 30)
                        return NO; // 30%
                else if (prob < 60)
                        return YES; // 30%
                else if (prob < 75)
                        return LATER; // 15%
                else if (prob < 98)
                        return SOON; // 13%
                else
                        return NEVER; // 2%
        }
}
class AskMe implements SharedConstants
{
        public static void main(String args[ ])
        {
                Question q = new Question();
                q.ask();
                q.ask();
                q.ask();
                q.ask();
        }
}
```

## Interfaces Can Be Extended

One interface can inherit another by use of the keyword extends.
Example:

```
interface A
{
        void meth1();
        void meth2();
}


// B now includes meth1() and meth2() — it adds meth3().
interface B extends A
{
        void meth3();
}


// This class must implement all of A and B
class MyClass implements B
{
        public void meth1( )
        {
                System.out.println("Implement meth1().");
        }

        public void meth2( )
        {
                System.out.println("Implement meth2().");
        }

        public void meth3( )
        {
                System.out.println("Implement meth3().");
        }
}

public class ex
{
        public static void main(String arg [ ])
        {
                MyClass ob = new MyClass();
                ob.meth1();
                ob.meth2();
                ob.meth3();
        }
}
```

Two more rules

1. A class can implement more than one interface but can extend only one class.
2. An interface can itself extend another interface, but never implement anything. An interface can extend more than one interface.

Other differences

1. Class has state , Interface does not (all identifiers are constants)
2. Class members can be private, protected, public, whereas the interface members cannot be private & protected.
3. Classes can be instantiated (via objects). Interface cannot be instantiated.
4. Interface slow the program as compared to classes.

(S1/T2/Q2). Write a JAVA Program to illustrate the concept of multiple interface implementation. Also illustrate how an interface can be extended.
Ans:

```
interface A
{
        void disp1();
}

interface B
{
        void disp2();
}

interface C extends A           //inherits disp1 – demonstrates interface inheritance
{
        void disp3();
}

class D implements B,C          //demonstrating multiple interface implementation
{
        public void disp1( )            //implementation of inherited disp1 of class C
        {
                System.out.println("1");
        }
        public void disp2( )    //implementation of disp2 of class B
        {
                System.out.println("2");
        }
        public void disp3( )    //implementation of disp3 of class C
        {
                System.out.println("3");
        }
        public void disp4( )    //member function of class D
        {
                System.out.println("4");
        }
}
```

```
public class set1_t2_ex2
{
        public static void main(String [ ] args)
        {
                D d=new D();
                d.disp1();
                d.disp2();
                d.disp3();
                d.disp4();
        }
}
```

Output:
1
2
3
4

Give a brief explanation of the above program…….

(S1/T3/Q2). Write a JAVA Program to illustrate multiple catch statements after a try block.
Ans:

```
import java.io.*;

public class set1_t3_ex2
{
	public static void main(String args[ ])
	{
	BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
	System.out.println("Enter 2 numbers");

		try
		{
			double x=Double.parseDouble(br.readLine());
			double y=Double.parseDouble(br.readLine());
			double z=x/y;
			if (y==0) throw new ArithmeticException();
			System.out.println("z="+z);
		}
		catch(IOException e)				//aroused if readLine fails
		{
			System.out.println("Critical failure... readLine failed!!!");
		}
		catch(NumberFormatException e)		//aroused if parsing fails
		{
			System.out.println("You did not enter numeric input");
		}
		catch(ArithmeticException e)				//Aroused if division fails
		{
			System.out.println("Division by Zero error");
		}

	System.out.println(“Bye..”);
	}
}
```

The possible problems that could arise are

- IO Exception, when br.readLine() fails [IOException]
- x and y when entered are not numeric [NumberFormatException]
- z=x/y & y is 0 [Note if y=0 & x≠0 then z=Infinity & if x=0 and y=0 then z=NaN. Note in both these situation no catch handler is needed]. To have my own message of 'Division by Zero error' I had to throw an exception when y is 0, so that it will enter the corresponding catch].

The first catch to enter corresponds to the first exception that arises (if any) within the try. Once an exception has occurred (if any), the statements below that statement in the try block is skipped and the corresponding catch handler is excuted & the program proceeds to the statement(s) after the catch handlers[In this case it prints Bye].

(S1/T3/Q3). What is a thread? What is meant by multitasking / multithreading?
Ans:

Multitasking is performing two or more tasks at the same time. Nearly all operating systems are capable of multitasking by using one of two multitasking techniques: process-based multitasking and thread-based multitasking. The objective of multitasking is to utilize the idle time of the CPU.

Thread-based multitasking is having a program perform two tasks at the same time. For example, a word processing program can check the spelling of words in a document while you write the document. This is thread-based multitasking.

The difference between process-based multitasking and thread-based multitasking is to think of process-based as working with multiple programs and thread-based as working with parts of one program.

Process-based multitasking has a larger overhead than thread-based multitasking since the threads in thread-based multitasking share the same address space in memory because they share the same program and switching from one part of the program to another happens within the same address space in memory.

A thread is part of a program that is running. Thread-based multitasking has multiple threads running at the same time (that is, multiple parts of a program running concurrently). Each thread is a different path of execution.

Example: Let's return to the word processing program example to see how threads are used. Two parts of the word processor are of interest: The first is the part of the program that receives characters from the keyboard, saves them in memory, and displays them on the screen. The second part is the portion of the program that checks spelling. Each part is a thread that executes independently of each other, even though they are part of the same program. While one thread receives and processes characters entered into the keyboard, the other thread sleeps. That is, the other thread pauses until the CPU is idle. The CPU is normally idle between keystrokes. It is this time period when the spell checker thread awakens and continues to check the spelling of the document. The spell checker thread once again pauses when the next character is entered into the keyboard.

The Java run-time environment manages threads.
A thread can be in one of four states:

- Running A thread is being executed.
- Suspended Execution is paused and can be resumed where it left off.
- Blocked A resource cannot be accessed because it is being used by another thread.
- Terminated Execution is stopped and cannot be resumed.

[Draw the diagram of thread life cycle…..]

All threads are not equal. Some threads are more important than other threads and are giving higher priority to resources such as the CPU. Each thread is assigned a thread priority that is used to determine when to switch from one executing thread to another. This is called *context switching*.

A thread's priority is relative to the priority of other threads. That is, a thread's priority is irrelevant if it is the only thread that is running. A lower-priority thread runs just as fast as a higher-priority thread if no other threads are executing concurrently.

Thread priorities are used when the rules of context switching are being applied. These rules are as follows:

- A thread can voluntarily yield to another thread. In doing so, control is turned over to the highest-priority thread.
- A higher-priority thread can preempt a lower-priority thread for use of the CPU. The lower-priority thread is paused regardless of what it's doing to give way to the higher-priority thread. Programmers call this *preemptive multitasking*.
- Threads of equal priority are processed based on the rules of the operating system.

Synchronization

Multithreading occurs *asynchronously*, meaning one thread executes independently of the other threads. In this way, threads don't depend on each other's execution. Sometimes the execution of a thread is dependent on the execution of another thread. Java enables you to synchronize threads by defining a synchronized method. A thread that is inside a synchronized method prevents any other thread from calling another synchronized method on the same object.

(S1/T3/Q5). What are the two methods of creating threads?
Ans:

The two methods of creating threads is by extending the 'Thread' class or by implementing the 'Runnable' interface.

If the class is alreading extending some class then we have no option but to resort to implementing from the Runnable interface, since JAVA does not support multiple inheritance. [Note that any class can implemement any number of interfaces].

Hence,

```
class C extends A,Thread                    //INVALID
{
        ……….
}

class C extends A implement B,Runnable      //VALID
{
        ……….
}
```

Method 1:

```
class MyThread extends Thread
{
        public void run( )
        {
                for(int i=1;i<1000;i++)
                        System.out.println(i);
        }
}

public class s1_t3_ex5a
{
        public static void main(String args[ ])
        {
                MyThread A=new MyThread( );
                MyThread B=new MyThread( );
                A.start( );
                B.start( );
        }
}
```

Method 2:

```
class MyThread implements Runnable
{
        public MyThread( )
        {
                Thread t=new Thread(this);
                t.start( );
        }

        public void run( )
        {
                for(int i=1;i<1000;i++)
                        System.out.println(i);
        }
}
public class s1_t3_ex5b
{
        public static void main(String args[ ])
        {
                MyThread A=new MyThread( );
                MyThread B=new MyThread( );
        }
}
```

(S1/T3/Q6). What is synchronization? When do we use it?
Ans:

Multithreading occurs *asynchronously*, meaning one thread executes independently of the other threads. In this way, threads don't depend on each other's execution. Sometimes the execution of a thread is dependent on the execution of another thread. Java enables you to synchronize threads by defining a synchronized method. A thread that is inside a synchronized method prevents any other thread from calling another synchronized method on the same object.

Imagine the havoc that can occur when two different threads have access to a single instance of a class, and both threads invoke methods on that object…and those methods modify the state of the object? A scenario like this might corrupt an object's state (by changing its instance variable values in an inconsistent way), and if that object's state is data shared by other parts of the program, well, it's too scary to even visualize.

Scary Example: Imagine a couple, Fred and Lucy, who both have access to the account and want to make withdrawals. But they don't want the account to ever be overdrawn, so just before one of them makes a withdrawal, he or she will first check the balance to be certain there's enough to cover the withdrawal.
Also, withdrawals are always limited to an amount of 10, so there must be at least 10 in the account balance in order to make a withdrawal. Sounds reasonable. But that's a two-step process:
1. Check the balance.
2. If there's enough in the account (in this example, at least 10), make the withdrawal.
What happens if something separates step 1 from step 2? For example, imagine what would happen if Lucy checks the balance and sees that there's just exactly enough in the account, 10. *But before she makes the withdrawal, Fred checks thebalance and also sees that there's enough for his withdrawal.* Since Lucy has verified the balance, but not yet made her withdrawal, Fred is seeing "bad data." He is seeing the
account balance *before* Lucy actually debits the account, but at this point that debit is certain to occur. Now both Lucy and Fred believe there's enough to make their withdrawals. So now imagine that Lucy makes *her* withdrawal, and now there isn't enough in the account for Fred's withdrawal, but he thinks there is since when he checked, there was enough! Yikes.

<u>Key points about locking and synchronization:</u>
■ Only *methods* can be synchronized, not variables.
■ Each object has just *one* lock.
■ *Not all methods in a class must be synchronized*. A class can have both synchronized and nonsynchronized methods.
■ If two methods are synchronized in a class, only one thread can be accessing one of the two methods. In other words, once a thread acquires the lock on an object, no other thread can enter *any* of the synchronized methods in that class (for that object).
■ If a class has both synchronized and nonsynchronized methods, *multiple threads can still access the nonsynchronized methods* of the class!
■ *If a thread goes to sleep, it takes its locks with it.*
■ *A thread can acquire more than one lock.*
■ *You can synchronize a block of code rather than a method*

Example:

```
class DISPLAY
{
        public synchronized void disp()
        {
                System.out.println();
                for(int i=1;i<100;i++)
                        System.out.print(i+",");
        }
}


class MyThread implements Runnable
{
        DISPLAY D;
        public MyThread(DISPLAY M)
        {
                D=M;                                    //NOTE: D is 'd' in case of both threads
                Thread t=new Thread(this);
                t.start( );
        }

        public void run( )
        {
                D.disp( );
        }
}
public class s1_t3_ex6
{
        public static void main(String args[ ])
        {
                DISPLAY d=new DISPLAY();
                MyThread A=new MyThread(d);             //NOTE: same object used, d
                MyThread B=new MyThread(d);     //NOTE: same object used, d
        }
}
```

Output:

```
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,
34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,6
3,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,
93,94,95,96,97,98,99,
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,
34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,6
3,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,
93,94,95,96,97,98,99,
```

Type 1:

6. Checked & unchecked exceptions
7. Explain the concept of garbage collection in Java. When is the garbage collector called?
8. Explain the terms over-loading and over-riding.
9. What are the differences in abstract classes and interfaces in java ? Explain
10. what is finalize( ) method in java ? Explain
11. What is the finally block in JAVA?
12. What is difference between throw & throws. Explain with an example.
13. What are Inner Classes?
14. What is dynamic method dispath in JAVA or how is runtime polymorphism achieved in JAVA

Type 2:

3. Differentiate: String and StringBuffer class.
4. Compare Vectors and arrays

Type 3:

7. Explain the Java terms super and this. What is the two uses of super.
8. What is polymorphism? Explain with an example
9. Explain the concept of packages in Java. How are they used?
10. How is multiple inheritance implemented in Java?
11. Write a Java program to illustrate the concept of constructor overloading.
12. Object class is a super class of all java classes. Explain

Type 4:

6. Explain pass-by-value and pass-by-reference. Write a short segment of code to demonstrate the difference.
7. Describe the life-cycle of a Java application.
8. Explain the concept of Just-in-time compilation
9. What is meant by Java Virtual Machine?
10. Explain the features of Java ?
11. Write a note on Wrapper classes in Java.
12. Explain all the standard data types available in Java.

(S2/T1/Q1). Checked and Unchecked Exception
Ans:
Simply we can say that all checked exceptions are those which the program has to handle via 'catch' or the program needs to 'throws' it so that JVM handles it. Unchecked exception may not be caught by the program & the programmer does not have to be mention it in 'throws'.

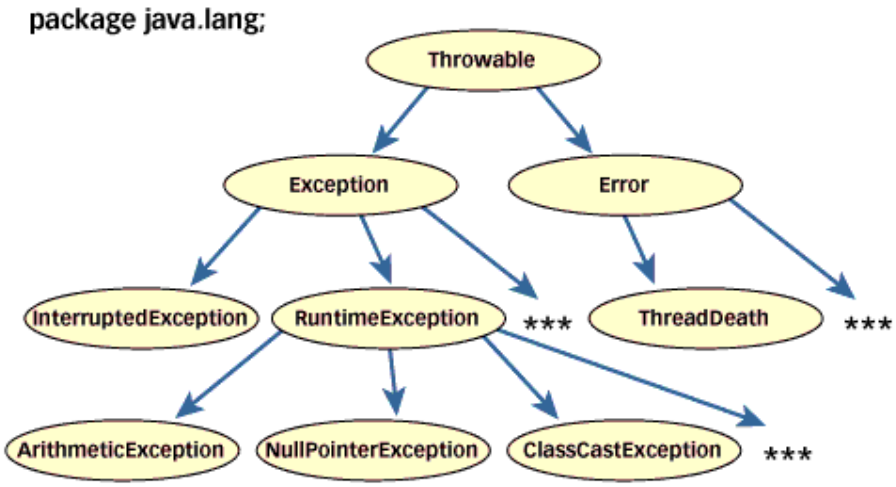



Figure 1. A partial view of the `Throwable` family

There are two kinds of exceptions in Java, *checked* and *unchecked*, and only checked exceptions need appear in throws clauses. The general rule is: Any checked exceptions that may be thrown in a method must either be caught or declared in the method's throws clause. Checked exceptions are so called because both the Java compiler and the Java virtual machine check to make sure this rule is obeyed.
Whether or not an exception is "checked" is determined by its position in the hierarchy of throwable classes. Figure 4, below, shows that some parts of the Throwable family tree contain checked exceptions while other parts contain unchecked exceptions. To create a new checked exception, you simply extend another checked exception. All throwables that are subclasses of Exception, but not subclasses of RuntimeException are checked exceptions.

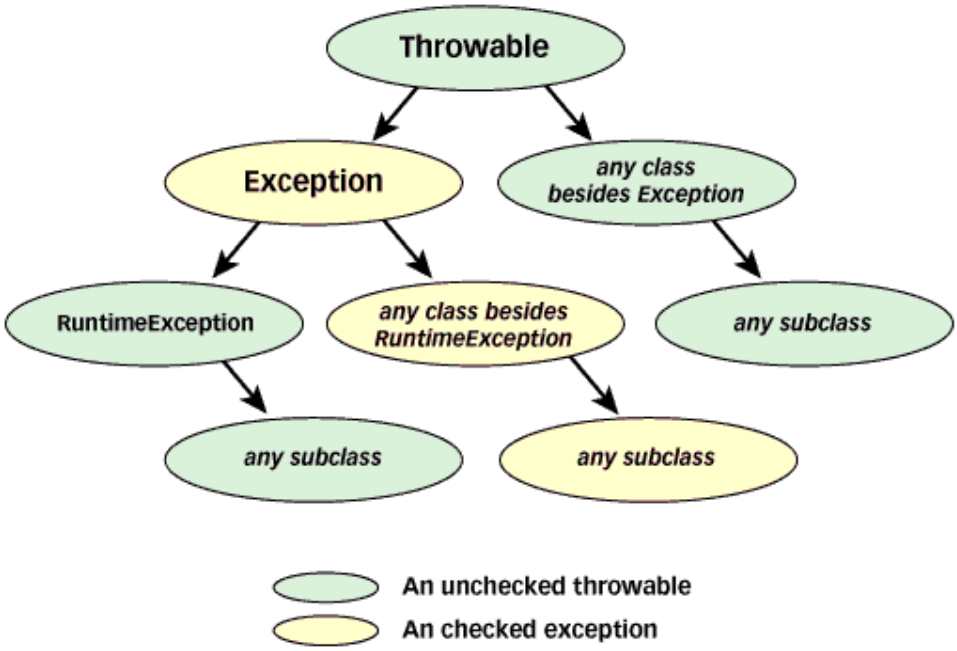



Figure 4. Checked and unchecked throwables

The conceptual difference between checked and unchecked exceptions is that checked exceptions signal abnormal conditions that you want client programmers to deal with.

When you place an exception in a throws clause, it *forces* client programmers/class who invoke the method to deal with the exception, either by catching it or by declaring it in their own throws clause. If they don't deal with the exception in one of these two ways, their classes won't compile.

Most unchecked throwables declared in java.lang (subclasses of Error and RuntimeException) are problems that would be detected by the Java virtual machine. Errors usually signal abnormal conditions that you wouldn't want a program to handle. In the rare cases in which they happen, it is usually reasonable that the thread terminate.

Problems with arrays, such as ArrayIndexOutOfBoundsException, or passed parameters, such as IllegalArgumentException, also could happen just about anywhere in a program. When exceptions like these are thrown, you'll want to fix the software bugs that caused them to be thrown. Java makers won't, however, want to force programmers to wrap every invocation of a method that uses arrays with a catch clause for ArrayIndexOutOfBoundsException. Hence some similar exceptions are 'Unchecked Exception'. You can throw and catch unchecked exceptions just like checked exceptions, but don't need to do so.

(S2/T1/Q2). Explain the concept of garbage collection in JAVA. When is the garbage collector called?
Ans:

Since objects are dynamically allocated by using the new operator, you might be wondering how such objects are destroyed and their memory released for later reallocation. In some languages, such as C++, dynamically allocated objects must be manually released by use of a delete operator. Java takes a different approach; it handles deallocation for you automatically. The technique that accomplishes this is called *garbage collection.* It works like this: when no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed. There is no explicit need to destroy objects as in C++. Garbage collection only occurs sporadically/occasionally (if at all) during the execution of your program. It does not execute every time one or more objects exist that are no longer used. You generally do not have to think about garbage collection while writing your programs.

Garbage collection describes automatic memory management in Java. Whenever a software program executes it uses memory in several different ways. The *heap* is that part of memory where Java objects live, and it's the one and only part of memory that is in any way involved in the garbage collection process. So, all of garbage collection revolves around making sure that the heap has as much free space as possible.

When the garbage collector runs, its purpose is to find and delete objects that cannot be reached. If you think of a Java program as in a constant cycle of creating the objects it needs (which occupy space on the heap), and then discarding them when they're no longer needed, creating new objects, discarding them, and so on, the missing piece of the puzzle is the garbage collector. When it runs, it looks for those discarded objects and deletes them from memory so that the cycle of using memory and releasing it can continue.

When Does the Garbage Collector Run?
The garbage collector is under the control of the JVM. The JVM decides when to run the garbage collector. From within your Java program you can ask the JVM to run the garbage collector, but there are no guarantees, under any circumstances, that the JVM will comply. Left to its own, the JVM will typically run the garbage collector when it senses that memory is running low.

Example 1 (null an object to make it eligible for garbage collection):

```
public class ex1
{
        public static void main(String [ ] args)
        {
                StringBuffer sb = new StringBuffer("hello");
                System.out.println(sb);
                // The StringBuffer object is not eligible for collection
                sb = null;
                // Now the StringBuffer object is eligible for collection
        }
}
```

Example 2 (reassigning a new value to an object reference to make it eligible for garbage collection):

```
public class ex2
{
        public static void main(String [ ] args)
        {
                StringBuffer s1 = new StringBuffer("hello");
                StringBuffer s2 = new StringBuffer("goodbye");
                System.out.println(s1);
                // At this point the StringBuffer "hello" is not eligible
                s1 = s2; // Redirects s1 to refer to the "goodbye" object
                // Now the StringBuffer "hello" is eligible for collection
        }
}
```

Interfaces vs Abstract Classes

| feature | Interface | abstract class |
|---|---|---|
| multiple inheritance | A class may implement several interfaces. | A class may extend only one abstract class. |
| default implementation | An interface cannot provide any code at all | An abstract class can provide abstract function signatures and some non-abstract methods with the code. |
| constants | Static final constants only, can use them without modification in classes that implement the interface. | Both instance/variable and 'static constants' are possible. Both static and instance intialising code are also possible. |
| third party convenience | An interface implementation may be added to any existing third party class. | A third party class must be rewritten to extend ONLY from the abstract class, since it cannot extend more than one class. |
| is-a vs -able or can-do | Interfaces are often used to describe the peripheral abilities of a class, not its central identity, e.g. an Automobile class might implement the Recyclable interface, which could apply to many otherwise totally unrelated objects. | An abstract class defines the core identity of its descendants. If you defined a Dog abstract class then Dalmatian descendants are Dogs. |
| plug-in | You can write a new replacement module for an interface (the implementation) that contains not one stick of code in common with the existing implementations. When you implement the interface, you start from scratch without any default implementation. Nothing comes with the interface other than a few constants. This gives you freedom to implement a radically different internal design. | You must use the abstract class as-is for the code base, with all its attendant baggage(other functions), good or bad. The abstract class author has imposed structure on you. |
| homogeneity | If all the various implementations share is the method signatures, then an interface works best. | If the various implementations are all of a kind and share a common status and behaviour, usually an abstract class works best. Another issue that's important is what I call "heterogeneous vs. homogeneous." If implementors/subclasses are homogeneous, tend towards an abstract base class. If they are heterogeneous, use an interface. If the various objects are all of-a-kind, and share a common |

Interfaces vs Abstract Classes

| feature | Interface | abstract class |
|---|---|---|
| | | state and behavior, then tend towards a common base class(abstract class). If all they share is a set of method signatures, then tend towards an interface. |
| speed | Slow, requires extra indirection to find the corresponding method in the actual class. Modern JVMs are discovering ways to reduce this speed penalty. | Fast |
| terseness | The constant declarations in an interface are all presumed public static final, so you may leave that part out. You can't call any methods to compute the initial values of your constants. You need NOT declare individual methods of an interface abstract. They are all presumed so. | You can put shared code into an abstract class, where you cannot into an interface. You may use methods to compute the initial values of your constants and variables, both instance and static. You MUST declare all the individual methods of an abstract class abstract (which are to be abstract). |
| adding functionality | If you add a new method to an interface, you must track down all implementations of that interface in the universe and provide them with a concrete implementation of that method. Means the new method will have to be defined by ALL the implementing classes. | If you add a new method to an abstract class, you have the option of providing a default implementation of it (in the abstract class). Then all existing code will continue to work without change. |

Give one program of interface[set1_Type2_ans1.pdf] & one of of abstract classes[set1_Type1_ans4.pdf]

(S2/T1/Q5). What is finalize( ) method in JAVA? Explain.
Ans:

Sometimes an object will need to perform some action when it is destroyed. For example, if an object is holding some non-Java resource such as a file handle, then you might want to make sure these resources are freed before an object is destroyed. To handle such situations, Java provides a mechanism called *finalization.* By using finalization, you can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector.

To add a finalizer to a class, you simply define the finalize( ) method. The Java run time calls that method whenever it is about to recycle an object of that class. Inside the finalize( ) method you will specify those actions that must be performed before an object is destroyed. The garbage collector runs periodically, checking for objects that are no longer referenced by any running state or indirectly through other referenced objects. Right before an asset is freed, the Java run time calls the finalize( ) method on the object.

The finalize( ) method has this general form:

```
protected void finalize( )
{
// finalization code here
}
```

Here, the keyword protected is a specifier that prevents access to finalize( ) by code defined outside its class.

It is important to understand that finalize( ) is only called just prior to garbage collection. It is not called when an object goes out-of-scope. This means that you cannot know when—or even if—finalize( ) will be executed. Therefore, your program should not rely on finalize( ) for normal program operation.

(S2/T1/Q6). What is finally block in JAVA?
Ans:
Say there is some compulsory code to be execute even through an exception does occur, then one senario is as follows:

```
try
{
   ……..
}
 catch(……..)
 {
      ……
  }
//the compulsory code
```

Looks simple. But now take the following example:

```
class A
{
        public void calc(double x,double y) throws ArithmeticException
        {       if (y==0) throw new ArithmeticException();
                System.out.println(x/y);
                System.out.println("Compulsory Code!!");
        }
}
public class s2_t1_ex6a
{       public static void main(String args[ ])
        {       A a=new A();
                try
                {
                        a.calc(2.2,0);
                }
                catch(ArithmeticException e)
                {System.out.println("Divide by Zero Error");
                }
        }
}
```

In the above example the function creates an exception if y is 0 but it isn't caught by the function, but it is caught my the calling function. We normally do this by exclosing the function call in a try – catch block & the function mentions the 'throws' criteria stating that it is not going to handle this exception by it is passing over the exception handling job to the calling function. [Here since the ArithmeticException is not a checked Exception, hence even if you don't mention throws it will work].
Notice how the program control flows:
First the function call is made passing the arguments by value to x & y respectively. Now, if y is 0, which it is in this case; the function will throw an ArithmeticException & transfer the control back to the calling function where the corresponding catch will handle the issue.
Note once an exception occurs the next thing to execute is the corresponding catch block & incase there is anything in between will be skipped. Hence the output of the above program will be
Divide by Zero Error

Now say for instance that there is some code in the function which has to execute whatever be the situation, then the above method would fail. When will such a need arise? One such situation is when we open a file in the function & if we do not close it cleanly then it creates un-necessary memory leaks. Imagine we open the file in the start of the function & close it at the end, but an exception occurs in between, then what? The program control will skip the code & does the catch handler & the file will not be closed. There can be other situations too like these. The basic gist is that we may want to execute some code before we leave the function to enter the corresponding catch.

To solve the above issue we enclose that portion of code in a block labelled finally. This ensures that the portion of code will happen, irrespective of the exception occuring or not. The revised example is shown below:

```
class A
{
	public void calc(double x,double y) throws ArithmeticException
	{
		try
		{
			if (y==0) throw new ArithmeticException();
			System.out.println(x/y);
		}
		finally
		{
			System.out.println("Compulsory Code!!");
		}
	}
}
public class s2_t1_ex6b
{
	public static void main(String args[ ])
	{
		A a=new A();
		try
		{
			a.calc(2.2,0);
		}
		catch(ArithmeticException e)
		{
			System.out.println("Divide by Zero Error");
		}
	}
}
```

The output now is

```
Compulsory Code!!
Divide by Zero Error
```

finally can also be used for try's which don't have catch.
finally is always excuted when the try is exited (either after exception being generated or after try block completes –no catch executed).

(S2/T1/Q8). What are Inner Classes?
Ans:
Introducing Nested and Inner Classes

It is possible to define a class within another class; such classes are known as *nested classes*. The scope of a nested class is bounded by the scope of its enclosing class. Thus, if class B is defined within class A, then B is known to A, but not outside of A. A nested class has access to the members, including private members, of the class in which it is nested. However, the enclosing class does not have access to the members of the nested class.

There are two types of nested classes: *static* and *non-static*. A static nested class is one which has the static modifier applied. Because it is static, it must access the members of its enclosing class through an object. That is, it cannot refer to members of its enclosing class directly. Because of this restriction, static nested classes are seldom used.

The most important type of nested class is the *inner* class. An inner class is a non-static nested class. It has access to all of the variables and methods of its outer class and may refer to them directly in the same way that other non-static members of the outer class do. Thus, an inner class is fully within the scope of its enclosing class.

The following program illustrates how to define and use an inner class. The class named Outer has one instance variable named outer_x, one instance method named test( ), and defines one inner class called Inner.

```
class Outer
{
	int outer_x = 100;
	void test( )
	{
		Inner inner = new Inner();
		inner.display();
	}
// this is an inner class
	class Inner
	{
		void display()
		{
			System.out.println("display: outer_x = " + outer_x);
		}
	}
}
class InnerClassDemo
{
	public static void main(String args[ ])
	{
		Outer outer = new Outer();
		outer.test();
	}
}
```

Output from this application is shown here:

```
display: outer_x = 100
```

In the program, an inner class named Inner is defined within the scope of class Outer. Therefore, any code in class Inner can directly access the variable outer_x. An instance method named display( ) is defined inside Inner. This method displays outer_x on the standard output stream. The main( ) method of InnerClassDemo creates an instance of class Outer and invokes its test( ) method. That method creates an instance of class Inner and the display( ) method is called.

It is important to realize that class Inner is known only within the scope of class Outer. The Java compiler generates an error message if any code outside of class Outer attempts to instantiate class Inner. Generalizing, a nested class is no different than any other program element: it is known only within its enclosing scope.

As explained, an inner class has access to all of the members of its enclosing class, but the reverse is not true. Members of the inner class are known only within the scope of the inner class and may not be used by the outer class.

(S2/T3/Q5). Write a program to illustrate the concept of constructor overloading.
Ans:

```
class BOX
{
        private double l,b,h;

        public BOX( )                                   //default constructor
        {}

        public BOX(double s)                            //parameterised for cube
        {
                l=b=h=s;
        }

        public BOX(double x, double y, double z)        //parameterised for cuboid
        {
                l=x;
                b=y;
                h=z;
        }

        public void putdata()
        {
                System.out.println("Volume="+l*b*h);
        }
}
public class s2_t3_q5
{
        public static void main(String args[ ])
        {
                BOX A=new BOX( );               //invokes 1st constructor
                BOX B=new BOX(10);              //invokes 2nd constructor - cube
                BOX C=new BOX(10,1,2);          //invokes 3rd constructor - cuboid

                A.putdata();
                B.putdata();
                C.putdata();
        }
}
```

Output:

```
Volume=0.0
Volume=1000.0
Volume=20.0
```

----------Explain the program in short ----------

(S2/T3/Q6). Object class is a super class of all classes. Explain.
Ans:
The Object Class

There is one special class, Object, defined by Java. All other classes are subclasses of Object. That is, Object is a superclass of all other classes. This means that a reference variable of type Object can refer to an object of any other class. Also, since arrays are implemented as classes, a variable of type Object can also refer to any array.

Object defines the following methods, which means that they are available in every object.

| Method | Purpose |
|---|---|
| Object clone( ) | Creates a new object that is the same as the object being cloned. |
| boolean equals(Object *object*) | Determines whether one object is equal to another. |
| void finalize( ) | Called before an unused object is recycled. |
| Class getClass( ) | Obtains the class of an object at run time. |
| int hashCode( ) | Returns the hash code associated with the invoking object. |
| void notify( ) | Resumes execution of a thread waiting on the invoking object. |
| void notifyAll( ) | Resumes execution of all threads waiting on the invoking object. |
| String toString( ) | Returns a string that describes the object. |
| void wait()<br>void wait(long *illiseconds*)<br>void wait(long m*illiseconds*,<br>int *nanoseconds*) | Waits on another thread of execution. |

The methods getClass( ), notify( ), notifyAll( ), and wait( ) are declared as final. You may override the others.
Example:

```
 class A
{
        public void putdata()
        {
                System.out.println("hi");
        }
}

public class s2_t3_q6
{
        public static void main(String args[ ])
        {
        Object o=new A( );
        ((A)o).putdata( );      //casting o to type A, since Object superclass of all object
        }                       //hence we can use 'o' to reference obj. of A & access methods of A
}
```

(S2/T4/Q1). Explain pass – by – value & pass – by – reference. Write a short code to demonstrate the difference.

Ans:

In pass by value the argument to the function are copies of the original actual parameters. Any changes made to the formal arguments will not affect the originals, whereas in call by reference the formal parameters are references to the actual parameters & any changes made to formal parameter will directly affect the actual parameters.

In JAVA only array & objects as arguments fall under the category of call by reference.

Let us consider the example of swapping two numbers.

Method 1:

```
public class s2_t4_q1a
{
        public static void swap(double m,double n)
        {
                double t=m;
                m=n;
                n=t;
        }

        public static void main(String args[ ])
        {
                double a=10,b=5;
                swap(a,b);
                System.out.println(a);//stays 10
                System.out.println(b);//stays 5
        }
}
```

Here the values of a & b i.e. 10 & 5 are respectively copied into m & n. By interchanging m & n there is no effect of a & b. Hence the output will be:

10

5

Thus in class by value any changes made to the formal parameters will not affect the actual parameters.

Method 2:

```
class Exchange
{       private double x;
        public Exchange(double a)
        {
                x=a;
        }

        public void swap(Exchange E)
        {       double t=x;
                x=E.x;
                E.x=t;
        }
```

```
        public void putdata()
        {
                System.out.println(x);
        }
}

public class s2_t4_q1b
{
        public static void main(String args[ ])
        {
                Exchange A=new Exchange(10);
                Exchange B=new Exchange(5);

                A.swap(B);

                A.putdata();
                B.putdata();
        }
}
```

Here the contents of 2 objects are swapped and when object is an argument to a function the formal parameter (here E) refers to the actual parameter (here B) & any changes made will reflect in the original. Hence the output will be

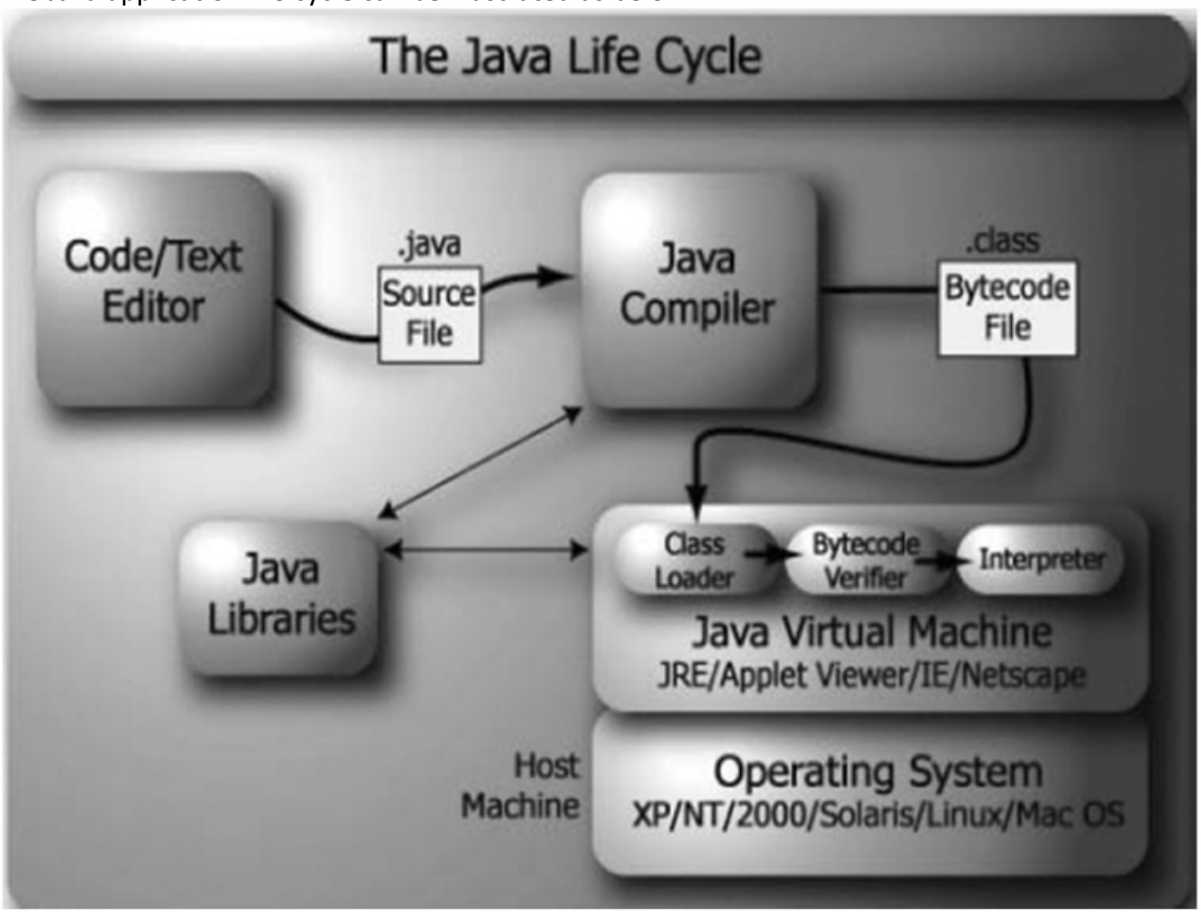
5

10

(S2/T4/Q2). Describe the life-cycle of a JAVA applcation.
Ans:

Java programs exist in the form of compiled bytecode, that is similar to machine code, except that the platform target is the Java Virtual Machine (JVM). A JVM resides within every Java compatible WWW browser and indeed stand-alone with the Java Run-time Environment (JRE).

A JVM is, in effect, a bytecode interpreting machine running on a hardware machine. This interpreting stage has an overhead and slows the program execution performance of Java applications. Java bytecode is extremely compact, allowing it to be easily delivered over a network. The Java compiler converts the .java source code to bytecode. At each stage the compiler and JVM interact with the Java library APIs (Application Programming Interface). The Java application life cycle can be illustrated as below:



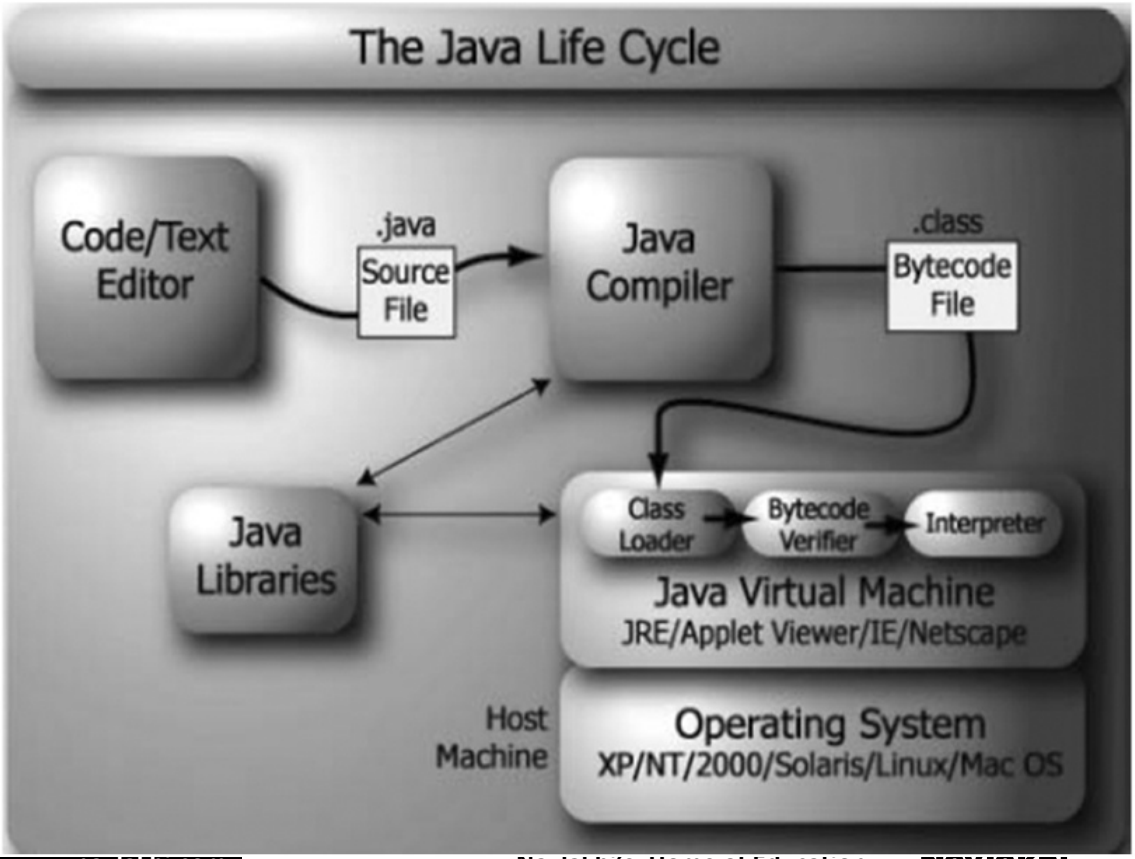

The figure above explains the lifecycle of a Java Program. In words, the figure can be explained as:

1. A Java program is written using either a Text Editor like Textpad or an IDE like Eclipse and is saved as a .java file. (Program.java)
2. The .java file is then compiled using Java compiler and a .class file is obtained from it. (Program.class)
3. The .class file is now portable and can be used to run this Java program in any platform.
4. Class file (Program.class) is interpreted by the JVM installed on a particular platform. JVM is part of the JRE software.

(S2/T4/Q4). What is meant by Java Virtual Machine?
Ans:
DEFINITION- A Java virtual machine (JVM), an implementation of the Java Virtual Machine Specification, interprets compiled Java binary code (called bytecode) for a computer's processor (or "hardware platform") so that it can perform a Java program's instructions. Java was designed to allow application programs to be built that could be run on any platform without having to be rewritten or recompiled by the programmer for each separate platform. A Java virtual machine makes this possible because it is aware of the specific instruction lengths and other particularities of the platform.

The Java Virtual Machine Specification defines an abstract -- rather than a real -- machine or processor. The Specification specifies an instruction set, a set of registers, a stack, a "garbage heap," and a method area. Once a Java virtual machine has been implemented for a given platform, any Java program (which, after compilation, is called bytecode) can run on that platform. A Java virtual machine can either interpret the bytecode one instruction at a time (mapping it to a real processor instruction) or the bytecode can be compiled further for the real processor using what is called a just-in-time compiler.

A JVM is, in effect, a bytecode interpreting machine running on a hardware machine. This interpreting stage has an overhead and slows the program execution performance of Java applications. Java bytecode is extremely compact, allowing it to be easily delivered over a network. The Java compiler converts the .java source code to bytecode. At each stage the compiler and JVM interact with the Java library APIs (Application Programming Interface). The Java application life cycle can be illustrated as below:



(S2/T4/Q5). Features of JAVA
Ans:
Although the fundamental forces that necessitated the invention of Java are portability and security The key considerations were summed up by the Java team as:

- Simple
- Secure
- Portable
- Object-oriented
- Robust
- Multithreaded
- Architecture-neutral
- Interpreted
- High performance
- Distributed
- Dynamic

### Simple

Java was designed to be easy for the professional programmer to learn and use effectively. Assuming that you have some programming experience, you will not find Java hard to master. If you already understand the basic concepts of object-oriented programming, learning Java will be even easier. Best of all, if you are an experienced C++ programmer, moving to Java will require very little effort. Because Java inherits the C/C++ syntax and many of the object-oriented features of C++, most programmers have little trouble learning Java. Also, some of the more confusing concepts from C++ are either left out of Java or implemented in a cleaner, more approachable manner.

### Object-Oriented

Although influenced by its predecessors, Java was not designed to be source-code compatible with any other language. This allowed the Java team the freedom to design with a blank slate. One outcome of this was a clean, usable, pragmatic approach to objects.

### Robust

The ability to create robust programs was given a high priority in the design of Java. To gain reliability, Java restricts you in a few key areas, to force you to find your mistakes early in program development. At the same time, Java frees you from having to worry about many of the most common causes of programming errors. Because Java is a strictly typed language, it checks your code at compile time. However, it also checks your code at run time. Knowing that what you have written will behave in a predictable way under diverse conditions is a key feature of Java. Memory management can be a difficult, tedious task in traditional programming environments. For example, in C/C++, the programmer must manually allocate and free all dynamic memory. This sometimes leads to problems, because programmers will either forget to free memory that has been previously allocated or, worse, try to free some memory that another part of their code is still using. Java virtually eliminates these problems by managing memory allocation and deallocation for you. Java provides garbage collection for unused objects. Exceptional conditions in traditional environments often arise in situations such as division by zero or "file not found," and they must be managed with clumsy and hard-to-read constructs. Java helps in this area by

providing object-oriented exception handling. In a well-written Java program, all run-time errors can—and should— be managed by your program.

### Multithreaded

Java was designed to meet the real-world requirement of creating interactive, networked programs. To accomplish this, Java supports multithreaded programming, which allows you to write programs that do many things simultaneously.

### Architecture-Neutral

A central issue for the Java designers was that of code longevity and portability. One of the main problems facing programmers is that no guarantee exists that if you write a program today, it will run tomorrow—even on the same machine. Operating system upgrades, processor upgrades, and changes in core system resources can all combine to make a program malfunction. The Java designers made several hard decisions in the Java language and the Java Virtual Machine in an attempt to alter this situation. Their goal was "write once; run anywhere, any time, forever." To a great extent, this goal was accomplished .

### Interpreted and High Performance

Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode. This code can be interpreted on any system that provides a Java Virtual Machine. Java was designed to perform well on very low-power CPUs. The Java bytecode was carefully designed so that it would be easy to translate directly into native machine code for very high performance by using a just-in-time compiler.

### Distributed

Java is designed for the distributed environment of the Internet, because it handles TCP/IP protocols. In fact, accessing a resource using a URL is not much different from accessing a file. Java has recently revived some interfaces in a package called *Remote Method Invocation* (*RMI*). This feature brings an unparalleled level of abstraction to client/server programming.

### Dynamic

Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time. This makes it possible to dynamically link code in a safe and expedient manner. This is crucial to the robustness of the applet environment, in which small fragments of bytecode may be dynamically updated on a running system.

JUST – IN- TIME (JIT) Compilation

In a bytecode-compiled system, source code is translated to an intermediate representation known as bytecode. Bytecode is not the machine code for any particular computer, and may be portable among computer architectures. The bytecode may then be interpreted by, or run on, a virtual machine. A just-in-time compiler can be used as a way to speed up execution of bytecode. At the time the bytecode is run, the just-in-time compiler will compile some or all of it to native machine code for better performance. This can be done per-file, per-function or even on any arbitrary code fragment; the code can be compiled when it is about to be executed (hence the name "just-in-time").
JIT code generally offers far better performance than interpreters. In addition, it can in some or many cases offer better performance than static compilation

1. The compilation can be optimized to the targeted CPU and the operating system model where the application runs.
2. The system is able to collect statistics about how the program is actually running and it can rearrange and recompile for optimum performance.
3. The system can do global code optimizations without the overheads of static compilers and linkers.

The just-in-time compiler comes with JVM and is used optionally. It compiles the bytecode into platform-specific executable code that is immediately executed.

Wrapper Class

The wrapper classes in the Java API serve two primary purposes:
■ To provide a mechanism to "wrap" primitive values in an object so that the primitives can be included in activities reserved for objects, like Vectors & Hastable
■ To provide functions for primitives. Most of these functions are related to various conversions: converting primitives to and from String objects.

There is a wrapper class for every primitive in Java. For instance the wrapper class for *int* is Integer, for *float* is Float, and so on. *Remember that the primitive name is simply the lowercase name of the wrapper except for char, which maps to Character, and int, which maps to Integer*.

| Primitive | Wrapper Class |
|---|---|
| boolean | Boolean |
| byte | Byte |
| char | Character |
| double | Double |
| float | Float |
| int | Integer |
| long | Long |
| short | Short |

All of the wrapper classes except Character provide two constructors: one that takes a primitive of the type being constructed, and one that takes a String representation of the type being constructed—for example,
Integer i1 = new Integer(42);
Integer i2 = new Integer("42");

WRITE ANY PROGRAM OF BufferedReader WITH PARSING……

Applet vs. Application

- Applet
  - Programs that are downloaded from the Web and executed inside of a browser
  - Cannot read or write to local file system
  - Cannot communicate with any server other than the one that the applet was downloaded from
  - Cannot run any program on user's system
  - Cannot load programs native to the local platform
- Application
  - Stand alone Java programs that do not require a Web browser to run
  - They are more general purpose programs
  - They do not have security restrictions that applets have
  - They are typically executed using the command line JVM (java)