KIIT POLYTECHNIC

# LECTURE NOTES

ON

# Object Oriented Methodology (JAVA)

3rd Semester, Computer Science & Engineering

Compiled by

**Er. Sunil Kumar Sahoo**
Lecturer in Department of Computer Science & Engineering
KIIT Polytechnic, Bhubaneswar
Email Id-sunilfcs@kp.kiit.ac.in

# CONTENTS

# OBJECT ORIENTED METHODOLOGY
# Unit-I

## PRINCIPLE OF OBJECT ORIENTED PROGRAMMING

**What is object oriented programming Language?**
Object Oriented Programming is a paradigm that provides many concepts such as

Object based features, inheritance, data binding and polymorphism

Example of object oriented programming languages:
- Simula.
- Smalltalk
- C++
- JAVA
- Python

**Basic concept OOPs (Object Oriented Programming System)**

- Object
- Class
- Data abstraction and encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message passing

### Object
- Instance of class is known as object.
- In other words objects are the basic runtime entities in an object-oriented system.
- Any entity that has state and behavior.
- For example: chair, pen, table, keyboard, bike etc are objects.
- It can be physical and logical.

### Class

- Collection of objects is known as class.
- It is a user defined data type
- It binds data member as well as member function together.

### Abstraction
- Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.
- In java, we use abstract class and interface to achieve abstraction.

### Encapsulation
- Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.
- A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

## Inheritance

- When one object acquires all the properties and behaviors of parent object i.e. known as inheritance.
- It provides code reusability. It is used to achieve runtime polymorphism.

## Benefits of Inheritance

- **Reusability** - facility to use public methods of base class without rewriting the same.
- **Extensibility** - extending the base class logic as per business logic of the derived class.

## Polymorphism

- Polymorphism is another important OOP concept.
- Polymorphism means the ability to take more than one form.
- That means one name multiple form.

There are two types of polymorphism

- **Compile time polymorphism**
- **Run time polymorphism**

## Dynamic Binding

- Binding refers to the linking of a procedure call to the code to be executed in response to the call.
- Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at runtime.

## Message Communication

- An object-oriented program consists of a set of objects that communicate with each other. The process of programming in an object-oriented language, therefore, involves the following basic steps:
- Creating classes that define objects and their behavior.
- Creating objects from class definitions,
- Establishing communication among objects.

**BENEFITS OF OOP:-**

- Code reusability is provided.
- Code duplication is avoided.
- Software complexity can be easily managed.
- Object oriented Systems can be easily upgraded from small to large systems.
- It is easy to partition the work in a project based on objects.

## PROGRAM PARADIGM

### Monolithic Programming language:-

- Data variables are declared globally and the statements are written in sequence.
- In this programming language, the concept of sub program does not exist.
- Examples: - BASIC, ASSEMBLY

### Procedural programming language:-

- Programs are divided into number of segments known as sub programs.
- Thus, it focuses the functions apart from data.
- Control is transferred using 'goto' statement.
- Data is global.
- All programs share the same data.
- Example: -C, FORTRAN, COBOL.

### Structured programming language: -

- Larger programs are developed, divided into multiple sub models and procedures.
- Each module has its own set of local and program codes.
- User defined data types are introduced in this programming language.
- Examples: - PASCAL, C

### Object Oriented Programming Language: -

- Problems are divided into objects.
- It is not possible to access data freely.
- Data hiding is possible.
- It supports bottom up approach in programming design.
- Object can exchange data through functions.

# UNIT-II

## Java Programming- History of Java

- Java is a general-purpose, true object-oriented programming language developed by Sun Microsystems of USA in 1991.
- Originally called Oak by James Gosling, one of the inventors of the language.
- It is the incremented version C and C++.
- That means JAVA inherits the properties of C and C++.

## Java Version History

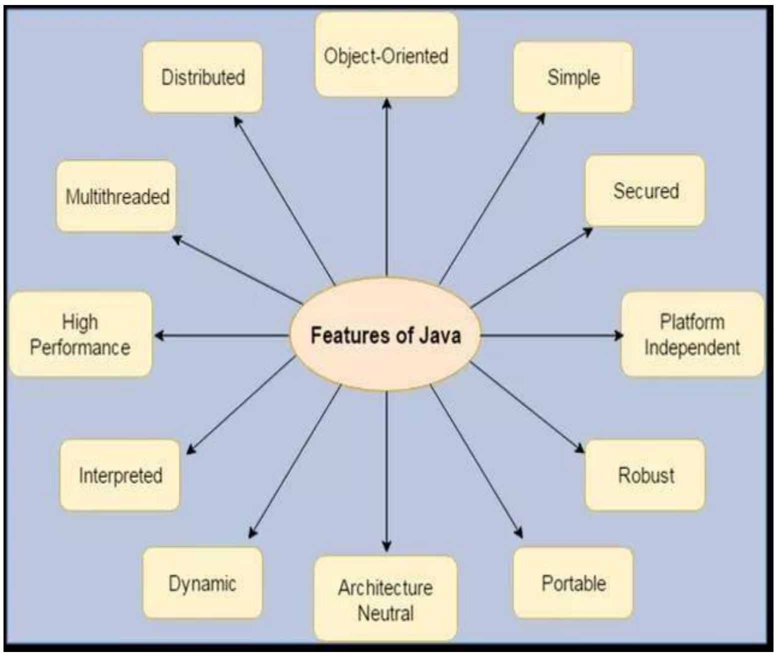
There are many java versions that has been released. Current stable release of Java is Java version jdk17

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan, 1996)
3. JDK 1.1 (19th Feb, 1997)
4. J2SE 1.2 (8th Dec, 1998)
5. J2SE 1.3 (8th May, 2000)
6. J2SE 1.4 (6th Feb, 2002)
7. J2SE 5.0 (30th Sep, 2004)
8. Java SE 6 (11th Dec, 2006)
9. Java SE 7 (28th July, 2011)
10. Java SE 8 (18th March, 2014)
11. Java version jdk17 (2021)

## Features of Java

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

1. Compile and interpreted
2. Object-Oriented
3. Portable
4. Platform independent
5. Distributed
6. Secured
7. Robust
8. Architecture neutral
9. Dynamic
10. High Performance
11. Multithreaded



**Compiled and Interpreted**

Usually a computer language is either compiled or interpreted. Java combines both these approaches thus making Java a two-stage system. First, Java compiler translates source code into what is known as byte code instructions. Byte codes are not machine instructions and therefore, in the second stage, Java interpreter generates machine code that can be directly executed by the machine that is running the Java program. We can thus say that Java is both a compiled and an interpreted language

**Platform-Independent and Portable**

Java programs can be easily moved from one computer system to another, anywhere and anytime. Changes and upgrades in operating systems, processors and system resources will not force any changes in Java programs.

Java ensures portability in two ways. First, Java compiler generates byte code instructions that can be implemented on any machine. Secondly, the size of the primitive data types are machine independent

**Object-Oriented**

Java is a true object-oriented language. Almost everything in Java is an object. All program code and data reside within objects and classes.

**Distributed**

Java is designed as a distributed language for creating applications on networks. It has the ability to share both data and programs. Java applications can open and access remote objects on Internet as easily as they can do in a local system. This enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

**Simple, Small and Familiar**

Java is a small and simple language. Many features of C and C++ that are either redundant or sources of unreliable code are not part of Java. For example, Java does not use pointers, preprocessor header files, goto statement and many others.

**Multithreaded and Interactive**

Multithreaded means handling multiple tasks simultaneously. Java supports multithreaded programs. This means that we need not wait for the application to finish one task before beginning another.

**High Performance**

Java performance is impressive for an interpreted language, mainly due to the use of intermediate bytecode. According to Sun, Java speed is comparable to the native C/C++. Java architecture is also designed to reduce overheads during runtime.

**Dynamic and Extensible**

Java is a dynamic language. Java is capable of dynamically linking in new class libraries, methods, and objects. Java can also determine the type of class through a query, making it possible to either dynamically link or abort the program, depending on the response.

**Java Comments**

The java comments are statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code for specific time.

**Types of Java Comments**

There are 3 types of comments in java.

1. Single Line Comment
2. Multi Line Comment
3. Documentation Comment

**Java Single Line Comment**

The single line comment is used to comment only one line.

**Syntax:**

1. //This is single line comment

**Example:**

**public class** CommentExample1 {

```
public static void main(String[] args) { int i=10;//Here, i is a variable
System.out.println(i);
}
}
```

Output:

```
10
```

## Java Multi Line Comment

The multi line comment is used to comment multiple lines of code.
**Syntax:**

```
/* This is
multi line comment
*/
public class CommentExample2 {
public static void main(String[] args) {
/* Let's declare and print variable in java. */
int i=10;
System.out.println(i);
} }
```

Output:

```
10
```

### Java Documentation Comment

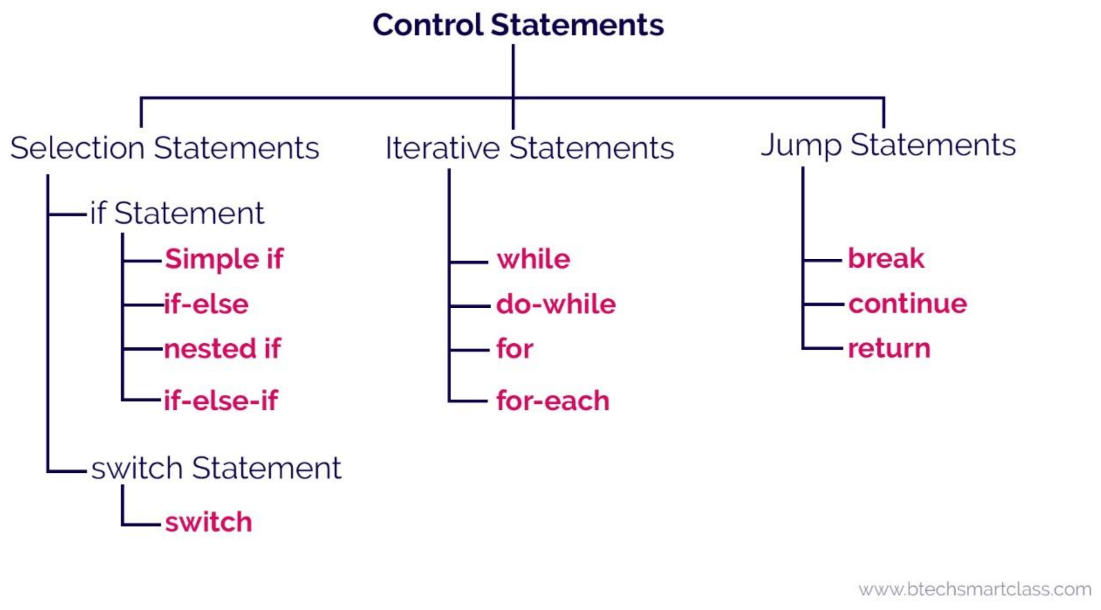
The documentation comment is used to create documentation API. To create documentation API, you need to use **javadoc tool**.

**Syntax:**

```
/** This is
documentation comment
*/
```

**Example:**

```
/** The Calculator class provides methods to get addition and subtraction of given 2 numbers.*/
```



**Types of Control Statements**

In java, the control statements are classified as follows.

- Selection Control Statements ( Decision Making Statements )
- Iterative Control Statements ( Looping Statements )
- Jump Statements

Let's look at each type of control statements in java.

**Selection Control Statements**

In java, the selection statements are also known as decision making statements or branching statements. The selection statements are used to select a part of the program to be executed based on a condition. Java provides the following selection statements.

- if statement – one way decision statement
- if-else statement-two way decision statement
- nested if statement- two way decision statement
- else if  ladder-multiway decision statement
- switch statement- multiway decision statement

### Iterative Control Statements

In java, the iterative statements are also known as looping statements or repetitive statements. The iterative statements are used to execute a part of the program repeatedly as long as the given condition is true. Using iterative statements reduces the size of the code, reduces the code complexity, makes it more efficient, and increases the execution speed. Java provides the following iterative statements.

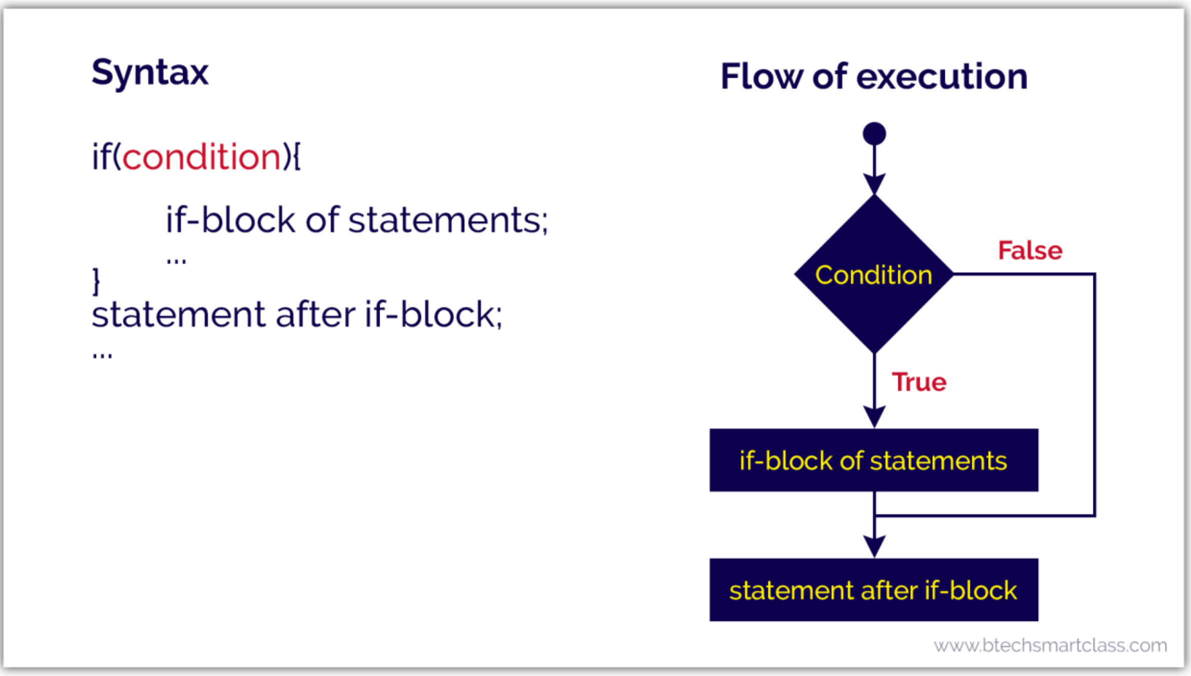
- while statement
- do-while statement
- for statement
- for-each statement

### Jump Statements

In java, the jump statements are used to terminate a block or take the execution control to the next iteration. Java provides the following jump statements.

- break
- continue
- return

## Syntax of if statement



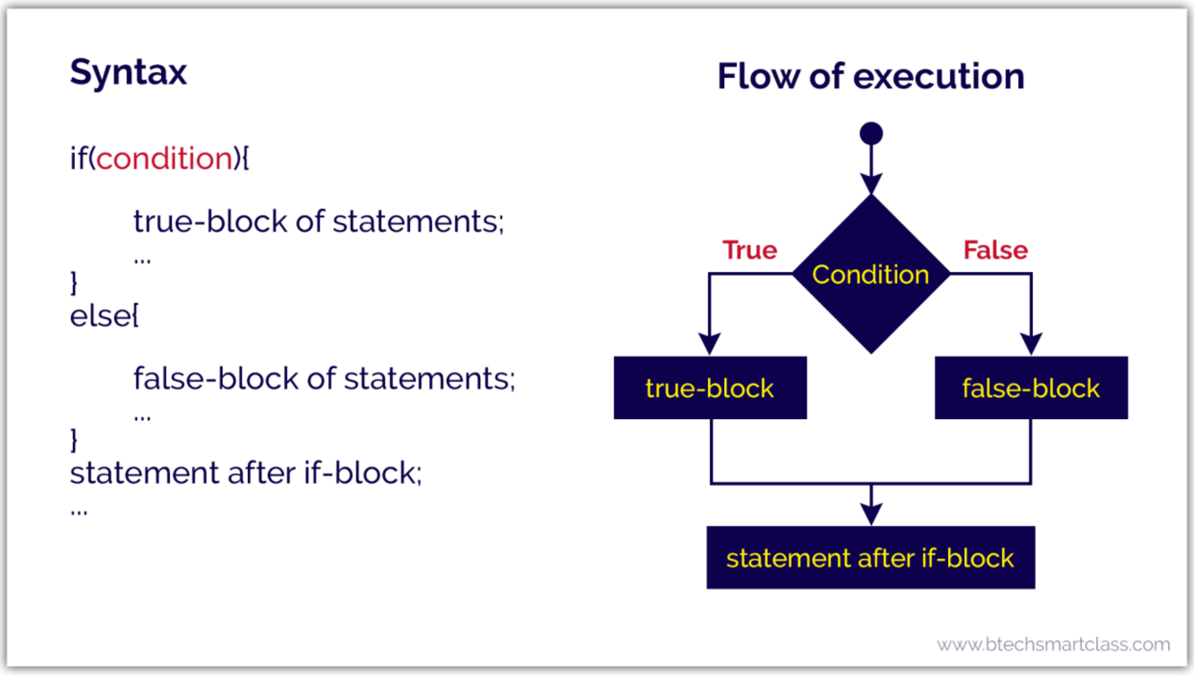
Syntax

```
if(condition){
        if-block of statements;
        ...
}
statement after if-block;
...
```

## Example of if statement

```
//WAP to input a number check whether it is divisible by 5 or not.
import java.util.Scanner;
public class A {

        public static void main(String args[]) {

          Scanner read = new Scanner(System.in);//it creates an object
          System.out.print("Enter any number: ");
          int num = read.nextInt();

          if((num % 5) == 0)
        {
                  System.out.println("Given number is divisible by 5!!");
          }

          System.out.println("We are outside the if-block!!!");

        }

}
```

## Syntax of if-else  statement



Syntax

```
if(condition){

        true-block of statements;
        ...
}
else{

        false-block of statements;
        ...
}
statement after if-block;
...
```

## Example of if –else statement

```
import java.util.Scanner;

public class B

{

        public static void main(String[] args)

{

         Scanner read = new Scanner(System.in);

         System.out.print("Enter any number: ");

         int num = read.nextInt();

         if((num % 2) == 0) {

                 System.out.println("We are inside the true-block!");

                 System.out.println("Given number is EVEN number!!");

         }

         else {

                 System.out.println("We are inside the false-block!");

                 System.out.println("Given number is ODD number!!");
```

```
            }
          }
}
```

## Syntax of if -else-if  statement

```
if(condition_1){
   condition_1 true-block;
   ...
}
else if(condition_2){
   condition_2 true-block;
     ...
}
else if(condition_3){
   condition_3 true-block;
     ...
}
else
      statement
```

## Example of else-if statement

```
//WAP in JAVA to find out greatest among three numbers using else if ladder
import java.util.Scanner;
public class C {

        public static void main(String[] args) {

          int num1, num2, num3;
          Scanner read = new Scanner(System.in);
          System.out.print("Enter any three numbers: ");
          num1 = read.nextInt();
          num2 = read.nextInt();
          num3 = read.nextInt();

          if( num1>=num2 && num1>=num3)
                  System.out.println("\nThe largest number is " + num1) ;

           else if (num2>=num1 && num2>=num3)
                  System.out.println("\nThe largest number is " + num2) ;

           else
                  System.out.println("\nThe largest number is " + num3) ;


        }

}
```

```
//WAP to input two numbers find out addition, subtraction, multiplication and division of two numbers
        using switch case

import java.util.Scanner;

class DD

{

        public static void main(String args[])

        {

          int num1,num2,opt,res;

          Scanner read = new Scanner(System.in);//it creates an object

          System.out.print("Enter first  number: ");

          num1 = read.nextInt();

          System.out.print("Enter second number: ");

          num2 = read.nextInt();

          System.out.println("1.add\n2.sub\n3.mul\n4.div: ");

          System.out.println("Enter your choice: ");

          opt = read.nextInt();

          switch(opt)

          {

                case 1:

                        res=num1+num2;

                        System.out.println("sum="+res);

                        break;

                case 2:

                        res=num1-num2;

                        System.out.println("sub="+res);

                        break;

                case 3:

                        res=num1*num2;

                        System.out.println("multi="+res);

                        break;
```

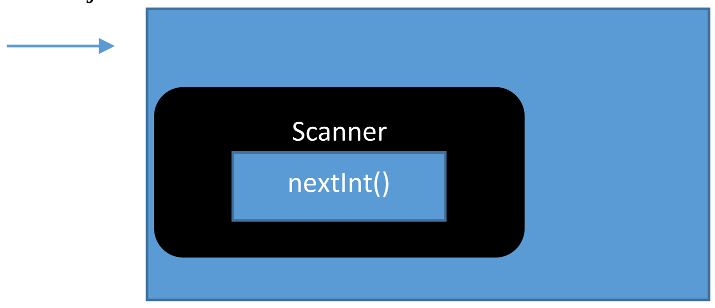
```
            case 4:
                        res=num1/num2;
                        System.out.println("div="+res);
                        break;
            default:
                        System.out.println("invalid input");
   }
}
```

Java.util.-package

Scanner-class

nextInt()-method

java.util



## What is loop?

The process of repeating a single statement or block of statements as long as the given condition is TRUE.

## Types of loop

i)while loop

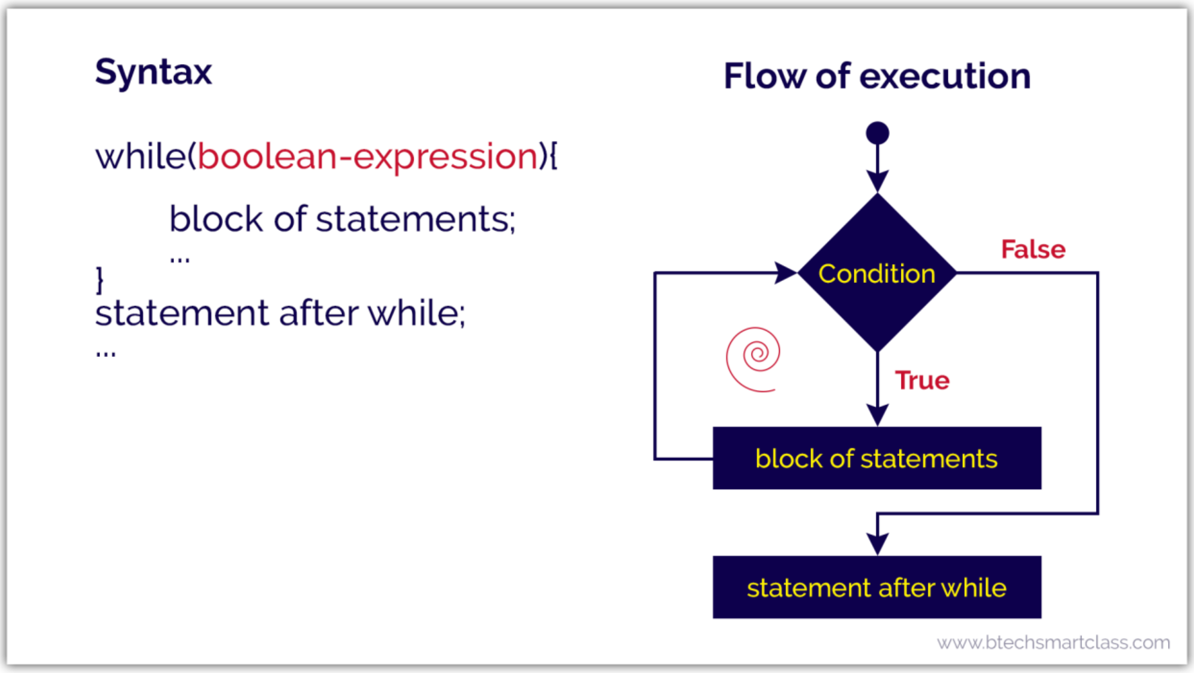
ii)for loop

iii)do while loop

iv)for each loop

## while loop in java

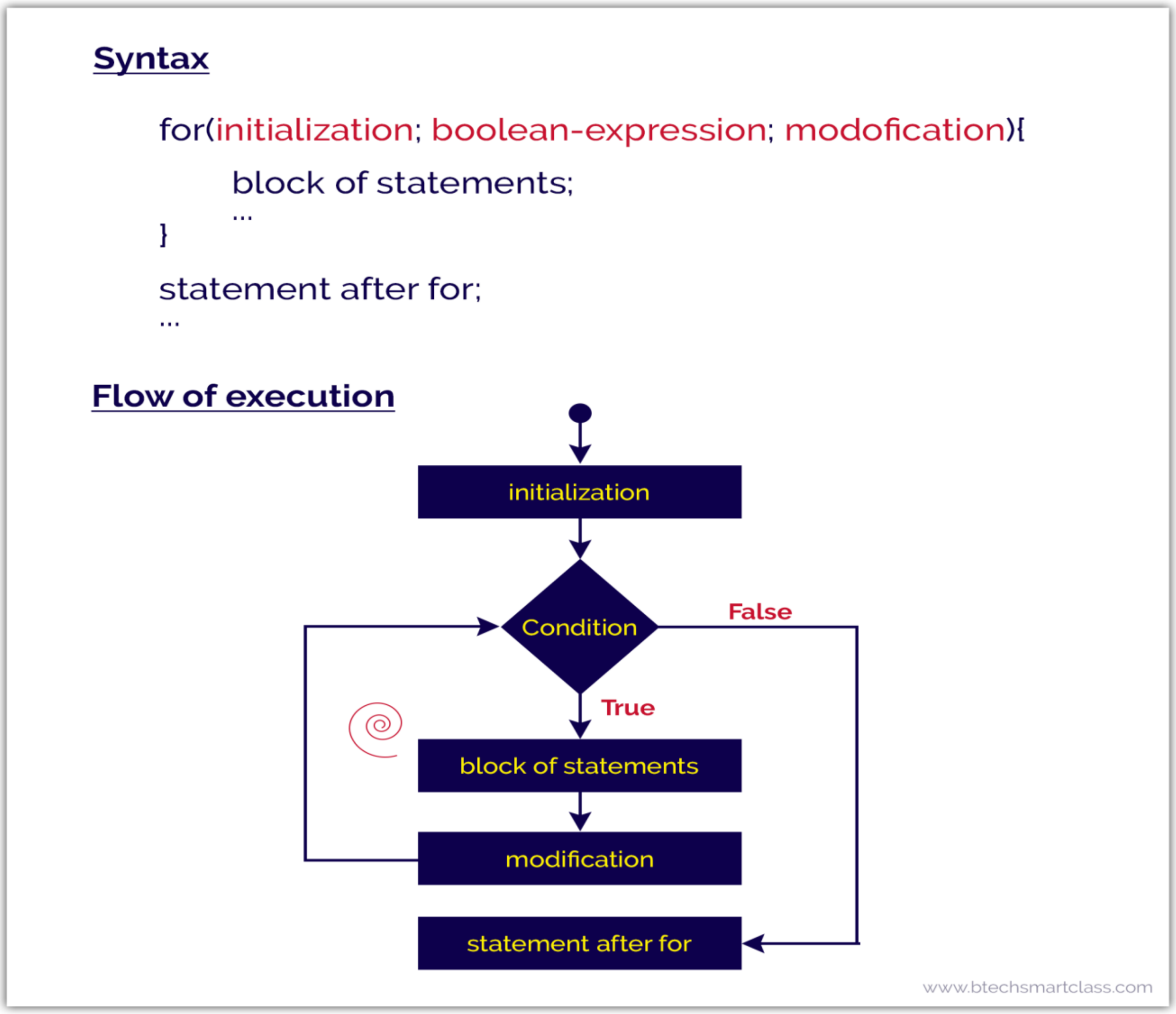
The while statement is used to execute a single statement or block of statements repeatedly as long as the given condition is TRUE. The while statement is also known as Entry control looping statement. The syntax and execution flow of while statement is as follows.



```
public class Test {

public static void main(String args[])

{

    int num = 1;
    while(num <= 10) {

     System.out.println(num);

      num++;

}

}

}
```

## for loop in java

The for statement is used to execute a single statement or a block of statements repeatedly as long as the given condition is TRUE. The for statement has the following syntax and execution flow diagram.



**Syntax**

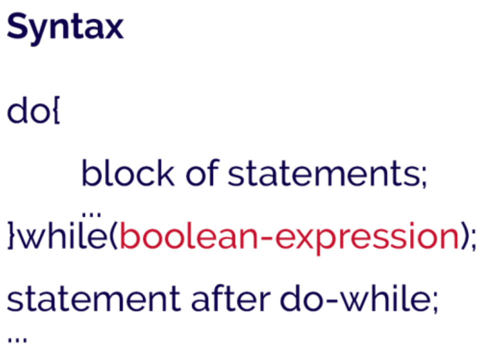
```
for(initialization; boolean-expression; modofication){
        block of statements;
        ...
}
statement after for;
...
```

**Flow of execution**

```
public class ForTest {


        public static void main(String[] args) {


                for(int i = 0; i < 10; i++) {

                        System.out.println("i = " + i);

                }


                System.out.println("Statement after for!");

        }


}

class k1

{

        public static void main(String args[])

        {

                int i=1;

              while(i<=3)

             {

                System.out.println("Hello World");

                i++;

              }

        System.out.println("Hello KP");

        }

}
```

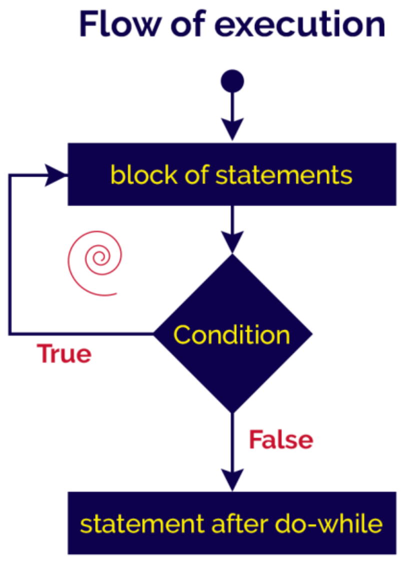
## do-while loop in java

The do-while statement is used to execute a single statement or block of statements repeatedly as long as given the condition is TRUE. The do-while statement is also known as the **Exit control looping statement**. The do-while statement has the following syntax.



**Syntax**

```
do{
      block of statements;
      ...
}while(boolean-expression);
statement after do-while;
...
```



```
public class DoWhileTest {

        public static void main(String[] args) {

                int num = 1;

                do {

                        System.out.println(num);

                        num++;

                   }while(num <= 10);

                     System.out.println("Statement after do-while!");
```

```
           } }
class k1
{
   public static void main(String args[])
  {

        int   i=1;
        do
         {

               System.out.println("Hello World");
                i++;
             }while(i<=3);
              System.out.println("Hello KP");
          }
}
```

OUTPUT:

```
Hello World
Hello World
Hello World
Hello KP
```

# Arrays

Normally, an array is a collection of similar type of elements which has contiguous memory location.

**Java array** is an object which contains elements of a similar data type. Additionally, the elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, and 2nd element is stored on 1st index and so on.

Unlike C/C++, we can get the length of the array using the length member. In C/C++, we need to use the sizeof operator.

In Java, array is an object of a dynamically generated class. Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces. We can store primitive values or objects in an array in Java. Like C/C++, we can also create single dimensional or multi-dimensional arrays in Java.

Moreover, Java provides the feature of anonymous arrays which is not available in C/C++.

Java provides a data structure, the **array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. This chapter introduces how to declare array variables, create arrays, and process arrays using indexed variables.

## Declaring and Initializing Array Variables:

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

```
dataType[] arrayRefVar; // preferred way. or
dataType arrayRefVar[]; // works but not preferred way.
dataType [] arrayRefVar = {value0, value1, ..., valuek};
```

Example:

The following is examples of above syntax:

Declaration and initialization

```
class arr
{
            public static void main(String args[])
            {

                        int k[]={10,20,30,40};
                        for(int i=0;i<k.length;i++)
                        {
                        System.out.print(" "+k[i]);
                        }
            }
}
```

Process
i=0<4 T
i=1<4 T
i=2<4 T
i=3<4 T

output

```
C:\kp>javac arr.java
C:\kp>java arr
 10 20 30 40
```

## Creating Arrays:

You can create an array by using the new operator with the following syntax:

```
DataType   arrayRefVar = new dataType[arraySize];
```

The above statement does two things:

- It creates an array using new dataType[arraySize];
- It assigns the reference of the newly created array to the variable arrayRefVar.

Example:
Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList:
 double []  x= new double[10];
Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.

**Processing Arrays:**
When processing array elements, we often use either for loop or for each loop because all of the elements in an array are of the same type and the size of the array is known.

Example:
Here is a complete example of showing how to create and process arrays:

```
        import java.util.Scanner;
        public class TestArray
        {
          public static void main(String[] args)
          {
               Scanner read =new Scanner(System.in);
             double[] x = new double[5];

            for (int i = 0; i < x.length; i++)
              {

               System.out.println( "enter the array elements  ");
              x[i]=read.nextDouble();
               }

        // Print all the array elements
          for (int i = 0; i < x.length; i++)
          {
          System.out.println(x[i] + " ");
          }
        // Sum of all elements
          double total = 0;
        for (int i = 0; i < x.length; i++)
        {
          total = total+x[i];
        }
        System.out.println("Total is " + total);
}
double avg=total/5;
System.out.println(avg);
}
```

This would produce the following result:

```
C:\kp>javac TestArray.java
C:\kp>java TestArray
enter the array elements
5.5
enter the array elements
4.4
enter the array elements
6.6
enter the array elements
7.7
enter the array elements
2.2

5.5
4.4
6.6
7.7
2.2
Total is 26.4
```

```
//2D array
import java.util.Scanner;
public class TestArray
{
public static void main(String[] args)
{
Scanner read =new Scanner(System.in);
int  x[][] = new int[3][3];

for (int i = 0; i < 3; i++)
{
  for (int j = 0; j < 3; j++)
  {
  System.out.println( "enter the array elements  ");
  x[i][j]=read.nextInt();
  }
}

// Print all the array elements
int total = 0;
for (int i = 0; i < 3; i++)
 {
  for (int j = 0; j < 3; j++)
  {

  System.out.print(x[i][j] + " ");
  total = total+x[i][j];
  }
  System.out.println( "\n");

}
// Sum of all elements
System.out.println("Total is " + total);
}
}
```

output

```
C:\kp>java TestArray
enter the array elements
1
enter the array elements
2
enter the array elements
3
enter the array elements
4
enter the array elements
5
enter the array elements
6
enter the array elements
7
enter the array elements
8
enter the array elements
9
1 2 3

4 5 6

7 8 9

Total is 45
```

# UNIT-III

## Java Classes and Objects

Java is an object-oriented programming language. Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake. A Class is like an object constructor, or a "blueprint" for creating objects.

**Create a Class**

To create a class, use the keyword **class**:

Create a class named "MyClass" with a variable x and method void disp()

```
 class MyClass {
              int x ;
             void disp()
            {
             x=20;
            System.out.println(x);
            }
        }
```

**Create an Object**

In Java, an object is created from a class. We have already created the class named MyClass, so now we can use this to create objects.

To create an object of MyClass, specify the class name, followed by the object name, and use the keyword new:

```
public class A{
  public static void main(String[] args) {
    MyClass Obj = new MyClass();
    Obj.disp();
  }
  }
```

**Example of classes and object**

```
class MyClass {
              int x ;
             void disp()
            {
             x=20;
            System.out.println(x);
            }
       }
public class A{
  public static void main(String[] args) {
    MyClass Obj = new MyClass();
    Obj.disp();
  }
 }
```

```
class Animal
{
             int x,y,z ;
             void sum()
            {
             x=20;
             y=30;
             z=x+y;
            System.out.println("sum="+z);
            }
}
public class A11
{
            public static void main(String[] args)
            {
            Animal lion = new Animal();
            lion.sum();
            }
}

// parameterized method
class Animal
{
             int x,y,z ;//data memeber/instance variable/attributes
             void sum(int a, int b)// method or function
            {
             x=a;
             y=b;
             z=x+y;
            System.out.println("sum="+z);
            }
}

public class A11
{
            public static void main(String[] args)
            {
            Animal lion = new Animal();
            lion.sum(10,20);
            }
}
```

**METHOD OVERLOADING**

/*Method overloading is a concept that allows to declare multiple methods with same name but different parameters in the same class. Method overloading is one of the ways through which java supports polymorphism. Polymorphism is a concept of object oriented programming that deal with multiple forms Method overloading can be done by changing number of arguments or by changing the data type of arguments. If two or more method have same name and same parameter list but differs in return type cannot be overloaded.*/

```
class Calculate
```

```
{
  void sum (int a, int b)
  {
    System.out.println("sum is"+(a+b)) ;
  }
  void sum (float a, float b)
  {
    System.out.println("sum is"+(a+b));
  }
}

class A15
{
  public static void main (String[] args)
  {
    Calculate  cal = new Calculate();
    cal.sum (8,5);      //sum(int a, int b) is method is called.
    cal.sum (4.6f, 3.8f); //sum(float a, float b) is called.
  }
}
```

**Find out area of the rectangle and circle using method overloading**

```
class Calculate
{
  int area(int l, int b)//Area of the rectangle
  {
    return(l*b);
  }
  double area (double r)// Area of the circle
  {
    return(3.141*r*r);
  }
}
```

```
class Ac1
{
  public static void main (String[] args)
  {
    Calculate  cal = new Calculate();
    int a1;
    double a2;
    a1=cal.area (8,5);
    System.out.println("the area of the rectangle="+a1);
   a2= cal.area (4.6);
   System.out.println("the area of the circle="+a2);
  }
}
```

# Unit-3

**CONSTRUCTOR:**

A constructor is a special member function whose task is to initialize the objects of its class. It is special because its name is the same as the class name. The constructor is invoked whenever an object of its associated class is created. It is called constructor because it construct the values of data members of the class.

The constructor functions have some characteristics:-

- They are invoked automatically when the objects are created.
- They don't have return types, not even void and therefore they cannot return values.
- They cannot be inherited, though a derived class can call.

**Types of Constructor**

1. Default constructor

The constructor having no arguments is known as default constructor.

```
class Calculate
{
        int x,y;
         Calculate ()
        {
        x=10;
         y=20;
         }
        void sum ()
         {
        System.out.println("sum is"+(x+y));
        }
}

class A16
{
         public static void main (String[] args)
        {
         Calculate  cal = new Calculate();
         cal.sum ();


         }
}
```

**2. Parameterized constructor**

The constructor having arguments is known as parameterized constructor.

```
class Calculate
{
           int x,y;
           Calculate (int a, int b)
             {
             x=a;
           y=b;
            }
            void sum ()
            {
            System.out.println("sum is"+(x+y));
            }
}

class A16
{
           public static void main (String[] args)
           {
           Calculate  cal = new Calculate(10,20);
             cal.sum ();
             }
}
```

**3. Constructor overloading-** Overloading refers to the use of same thing for different purposes.

When a class contains more than one constructor with different arguments is known as constructor overloading.

```
class Calculate
{
            int x,y;
            Calculate ()
             {
               x=100;
               y=200;
             }
             Calculate (int a, int b)
             {
               x=a;
               y=b;
             }
             void sum ()
             {
             System.out.println("sum is"+(x+y));
            }
}

class A16
{
  public static void main (String[] args)
  {
     Calculate  ob1 = new Calculate();
    Calculate  ob2= new Calculate(10,20);
    Ob1.sum ();
    Ob2.sum ();

   }
}
```

**4. Copy Constructor**

A copy constructor is used to declare and initialize an object from another object.

```
class code
{
            int id;
            code (int a)
            {
            id=a;
            }
            code(code x)
            {
            id=x.id;
            }
            void display( )
            {
             System.out.println(id);
            }
}




class A19
{
  public static void main (String[] args)
```

```
 {
        code ob1=new code(10);
        code ob2=new code(ob1);
        ob1.display();
        ob2.display();
}
}
```

## Difference between constructor and method in java

| Java Constructor | Java Method |
|---|---|
| Constructor is used to initialize the state of an object. | Method is used to expose behaviourof an object. |
| Constructor must not have return type. | Method must have return type. |
| Constructor is invoked implicitly. | Method is invoked explicitly. |
| The java compiler provides a default constructor if youdon't have any constructor. | Method is not provided by compiler inany case. |
| Constructor name must be same as the class name. | Method name may or may not be |

**Access Modifiers in java**

There are two types of modifiers in java: access modifiers and non-access modifiers.
The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.
There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

private access modifier
The private access modifier is accessible only within class.

Simple example of private access modifier
In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is compile time error.

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");} }
public class Simple{
public static void main(String args[]){ A obj=new A();
System.out.println(obj.data);//Compile Time Error obj.msg();//Compile Time Error
} }
```

2) default access modifier
If you don't use any modifier, it is treated as default bydefault. The default modifier is accessible only within package.
Example of default access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```
//save by A.java package pack; class A{
void msg(){System.out.println("Hello");}
}

//save by B.java package mypack; import pack.*;

class B{
public static void main(String args[])
{ A obj = new A();//Compile Time Error obj.msg();//Compile Time Error
}
}
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

3) protected access modifier

The protected access modifier is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```
//save by A.java package pack; public class A{
protected void msg(){System.out.println("Hello");} }
//save by B.java package mypack; import pack.*; class B extends A{
public static void main(String args[]){ B obj = new B();
obj.msg();
} }
 Output:Hello
```

4) public access modifier

The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

Example of public access modifier

```
 public class A{
public void msg(){System.out.println("Hello");} }
class B{
public static void main(String args[]){
 A obj = new A();
obj.msg();
} }
 Output:Hello
```

# UNIT-V

# INHERITANCE

- When one object acquires all the properties and behaviors of parent object i.e. known as inheritance.
- It provides code reusability. It is used to achieve runtime polymorphism.

**Benefits of Inheritance**

- **Reusability** - facility to use public methods of base class without rewriting the same.
- **Extensibility** - extending the base class logic as per business logic of the derived class.

**Syntax of Java Inheritance**

```
class Subclass-name extends Superclass-name

{

        //methods and fields

    }
```

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

***Terms used in Inheritance***

- ✓ **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- ✓ **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- ✓ **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

**Types of Inheritance based on the levels of inheritance and interrelation among the classes:**

- ✓ Single Inheritance
- ✓ Multiple Inheritance
- ✓ Hierarchical Inheritance
- ✓ Multilevel Inheritance
- ✓ Hybrid Inheritance
- ✓ Multipath Inheritance

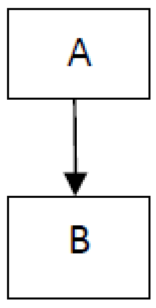
***Single Inheritance***: derivation of a class from only one base class.

***Multiple Inheritance***: derivation of a class from several (two or more) base classes.

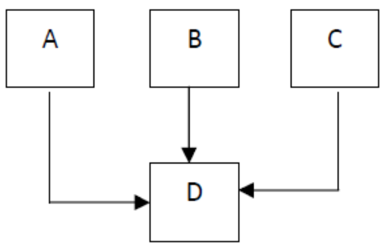
***Hierarchical Inheritance***: derivation of several classes from a single base class i.e. the traits of one class may be inherited by more than one class.

***Multilevel Inheritance***: derivation of a class from another derived class.

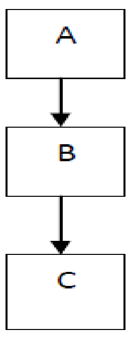
***Hybrid Inheritance***: derivation of a class involving more than one form of inheritance.



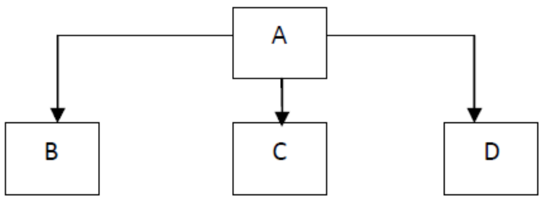
(Single inheritance)



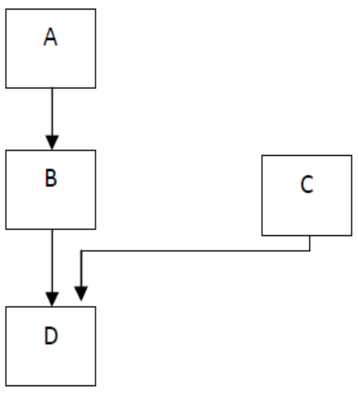
(Multiple Inheritance)



(Multilevel inheritance)



(Hierarchical inheritance)



**Hybrid Inheritance**

```
//Example  of Single  inheritance

class A

{

    int x,y;

    void input()

    {

     x=10;

     y=20;

    }

}

class B extends A

{


    int z;

    void output()

    {

       z=x+y;

     System.out.println("Sum="+z);

    }

}

class single

{

    public static void main(String args[])
    {
      B ob = new B();
      ob.input();
      ob.output();

    }
}
```

**Ouput-30**

```
class Animal{
  void eat()
{
  System.out.println("eating...");
  }
}
class Dog extends Animal{
void bark()
{
System.out.println("barking...");

}
}
class TestInheritance{
public static void main(String args[])
{
Dog d=new Dog();
d.bark();
d.eat();
}
}
```

Output:

```
barking... eating...
```

**// example of multilevel  heritance**

```
class A
{
   int a,b;//data member or instance varible
   void input()//method or function
   {
     a=10;
     b=20;
   }
}

class B extends A
{
   int c;
   void read()
   {
     c=30;
```

```
    }
}
class C extends B
{
    int r;
    void disp()
    {
            r=a+b+c;
            System.out.println(r);
    }
}
```

Output-60

**//Multilevel inheritance**

```
class A
{
    int x,y;
    void input()
    {
     x=10;
     y=20;
    }
}
class B extends A
{
    int z;
    void read()
    {
     z=30;
    }
}

class C extends B
{
      int r;
      void disp()
      {
      r=x+y+z;
     System.out.println("sum="+r);
     }
}
class demo6
{
    public static void main(String args[])
    {
      C ob = new C();
      ob.input();
      ob.read();
      ob.disp();
    }
```

```
}

class In
{
    public static void main(String args[])
    {
            C ob=new C();
            ob.input();
            ob.read();
            ob.disp();
    }
}
```

**//Hierarchical Inheritance**

```
class A
{
    int x,y;
    void input()
    {
            x=10;
            y=5;
    }
}
```

```
class B extends A
{
   int s=0;
   void sum()
   {

           s=x+y;
           System.out.println("sum="+s);
   }
}
class C extends A
{
   int p=0;
   void product()
   {
           p=x*y;
           System.out.println("product="+p);
   }
}

class Demo7
{
   public static void main(String args[])
   {
           B ob1 = new B();
           C ob2 = new C();
           ob1.input();
           ob1.sum();
           ob2.input();
           ob2.product();
   }
}
```

**Output**

```
Sum=15
Product=50
```

```
class Animal{
void eat()
{
System.out.println("eating...");
}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
```

```
public static void main(String args[]){ Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Output:

```
meowing... eating...
```

## super keyword in java

The **super** keyword in java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

**Usage of java super Keyword**

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

//SUPER KEYWORD IN  INHERITANCE

//case I- If the super class instance variable name is same as the child class instance variable then the super key word is used only inside the sub-class to access the super class instance variable.

```
class A
```

```
{
   int x=10;
   }
class B extends A
{
   int x=20;
   void show()
   {
           System.out.println(x);
           System.out.println(super.x);

   }
}
class S
{
   public static void main(String args[])
   {
           B ob1= new B();
           ob1.show();
   }
}
```

Output-

```
20

10
```

**Case-II**

JAVA uses super keyword for explicitly calling the super class constructor within the child class constructor.

```
Class A

{

   A ()

        {

   System.out.println ("hi");

}

}

Class B extends A

{

    B ()

{

   System.out.println("hellow");

}

}
```

```
Class Super1

{

     B ob=new B();

}
```

## Method Overriding

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

## Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

## Rules for Java Method Overriding

- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.
- There must be an IS-A relationship (inheritance)

```
class A

{

    void show()

    {
            System.out.println(" hello");
    }
}
class B extends A
{
    void show()
    {
            System.out.println("world!");
    }
}
class Demo9
{
    public static void main(String []args)
    {

    B ob=new B();
    ob.show();
    }
}
```

## final keyword

A final keyword can be used in 3 cases

Case 1- It can be used in a class. If the class is declared as final ,it cannot be inherited.

```
 final class A
{
    int x;
    void input()
    {
                x=10;
    }

}
class B extends A
{
    int z;
    void show()
    {
                z=x*x;
                System.out.println("value"+z);
    }

}


class Demo11

{

    public static void main(String args[])

    {

                B ob1 = new B();

                ob1.input();

                ob1.show();


    }

}
```

Output-Error

**Case- 2**

If the method is declared as final then it cannot be overridden*/

```
class A{

    final void show()

    {

    System.out.println("hello");
    }
}

class B extends A{
    void show()
    {
            System.out.println("world");
    }
}
class Demo12
{
    public static void main(String args[])
    {
            B ob1 = new B();
            ob1.show();
    }
}
```

Output-Error

**case-3**

If the variable is declared as final then it cannot be incremented or decremented*/

```
class Demo13{
        public static void main(String args[])
        {
        final int x=5;
        x++;
        System.out.println("result="+x);
    }
 }
```

Output-Error