JAVA

- Java is one of the programming language or technology used for developing web applications. Java language developed at SUN Micro Systems in the year 1995 under the guidance of James Gosling and there team.
- Whatever the software developed in the year 1990, SUN Micro Systems has released on the name of oak, which is original name of java (scientifically oak is one of the tree name). The OAK has taken 18 months to develop.

Example: Create First Java program

```
class FirstExamp
{
public static void main(String[] args)
{
System.out.println("Hello");
}
}
```

Output:

Save: FirstExamp.java

To compile: javac : FirstExamp.java

Run: java FirstExamp

Hello

Java divided into three categories, they are

- J2SE (Java 2 Standard Edition)
- J2EE (Java 2 Enterprise Edition)
- J2ME (Java 2 Micro or Mobile Edition)

1. **J2SE:** J2SE is used for developing client side applications.
2. **J2EE:** J2EE is used for developing server side applications.
3. **J2ME:** J2ME is used for developing mobile or wireless application by making use of a predefined protocol called WAP (wireless Access / Application protocol).

Define an API

- An **API (Application Programming Interface)** is a collection of packages, a package is the collection of classes, interfaces and sub-packages. A sub-package is a collection of classes, Interfaces and sub - packages etc.
- Java programming contains user friendly syntax so that we can develop effective applications. in other words if any language is providing user friendly syntax, we can develop error free applications.

Features of Java

- Features of a language are nothing but the set of services or facilities provided by the language vendors to the industry programmers. Some important **features of java** are

Important Features of Java

1. Simple
2. Platform Independent
3. Architectural Neutral
4. Portable
5. Multi Threading
6. Distributed
7. Networked
8. Robust
9. Dynamic
10. Secured
11. High Performance
12. Interpreted
13. Object Oriented

1. Simple

It is simple because of the following factors:

- It is **free from pointer** due to this execution time of application is improved. [Whenever we write a Java program without pointers then internally it is converted into the equivalent pointer program].
- It has **Rich set of API** (application protocol interface).

- It has **Garbage Collector** which is always used to collect un-Referenced (unused) Memory location for improving performance of a Java program.
- It contains user friendly syntax for developing any applications.

2. Platform Independent

- A program or technology is said to be platform independent if and only if which can run on all available operating systems with respect to its development and compilation. (Platform represents O.S)

3. Architectural Neutral: Architecture represents processor.

- A Language or Technology is said to be Architectural neutral which can run on any available processors in the real world without considering their development and compilation.

- The languages like C, CPP are treated as architectural dependent.

4. Portable

- If any language supports platform independent and architectural neutral feature known as portable. The languages like C, CPP, and Pascal are treated as non-portable language. It is a portable language. According to SUN Microsystems.

Portability = Platform Independent + Archecture

5. Multithreaded

- A flow of control is known as a thread. When any Language executes multiple thread at a time that language is known as multithreaded e. It is multithreaded.

6. Distributed

- Using this language, we can create distributed applications. RMI and EJB are used for creating distributed applications. In distributed application multiple client system depends on multiple server systems so that even problem occurred in one server will never be reflected on any client system.
- **Note:** In this architecture same application is distributed in multiple server system.

7. Networked

- It is mainly designed for web based applications; J2EE is used for developing network based applications.

8. Robust

- Simply means of Robust are strong. It is robust or strong Programming Language because of its capability to handle Run-time Error, automatic garbage collection, the lack of pointer concept, Exception Handling. All these points make It robust Language.

9. Dynamic

- It supports Dynamic memory allocation due to this memory wastage is reduce and improve performance of the application. The process of allocating the memory space to the input of the program at a run-time is known as dynamic memory allocation, to programming to allocate memory space by dynamically we use an operator called 'new
- 'new' operator is known as dynamic memory allocation operator.

10. Secure

- It is a more secure language compared to other language; In this language, all code is covered in byte code after compilation which is not readable by human.

11. High performance: It have high performance because of following reasons

- This language **uses Byte code** which is faster than ordinary pointer code so Performance of this language is high.
- **Garbage collector**, collect the unused memory space and improve the performance of the application.
- It has **no pointers** so that using this language we can develop an application very easily.
- It **supports multithreading**, because of this time consuming process can be reduced to executing the program.

12. Interpreted

- It is one of the highly interpreted programming languages.

13. Object Oriented

- It supports OOP's concepts because of this it is most secure language, for this topic you can read our oop's concepts in detail.

Object and class in Java

- Object is the physical as well as logical entity where as class is the only logical entity.

Class

- Class is a blue print which is containing only list of variables and method and no memory is allocated for them. A class is a group of objects that has common properties.

A class in java contains:

1. Data Member
2. Method
3. Constructor
4. Block
5. Class and Interface

Object: Object is an instance of class, object has state and behaviors.

An Object in java has three characteristics:

1. State
2. Behavior
3. Identity

1. **State:** Represents data (value) of an object.
2. **Behavior:** Represents the behavior (functionality) of an object such as deposit, withdraw etc.

3. **Identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.

- Class is also can be used to achieve user defined data types.

Real life example of object and class

- In real world many examples of object and class like dog, cat, and cow are belong to animal's class. Each object has state and behaviors. For example a dog has state: - color, name, height, age as well as behaviors:- barking, eating, and sleeping.

Vehicle class

- Car, bike, truck these all are belongs to vehicle class. These Objects have also different different states and behaviors. For Example car has state - color, name, model, speed, Mileage. as we; as behaviors - distance travel

Difference between Class and Object in Java

Class

1. Class is a container which collection of variables and methods.
2. No memory is allocated at the time of declaration
3. One class definition should exist only once in the program.

Object

1. object is a instance of class
2. Sufficient memory space will be allocated for all the variables of class at the time of declaration.
3. For one class multiple objects can be created.

Syntax to declare a Class

```
class Class_Name
 {
   Data member;
   Method;
 }
```

Simple Example of Object and Class

- In this example, we have created a Employee class that have two data members eid and ename. We are creating the object of the Employee class by new keyword and printing the objects value.

Example

```
class Employee
{
int eid;  // data member (or instance variable)
 String ename;  // data member (or instance variable)
 eid=101;
 ename="Vikash";
 public static void main(String args[])
 {
  Employee e=new Employee(); // Creating an object of class Employee
  System.out.println("Employee ID: "+e.eid);
  System.out.println("Name: "+e.ename);
}
}
```

Output

```
Employee ID: 101
Name: Vikash
```

Note: A new keyword is used to allocate memory at runtime, new keyword is used for create an object of class, later we discuss all the way for create an object of class

DATATYPE

- **Data type** is a special keyword used to allocate sufficient memory space for the data, in other words Data type is used for representing the data in main memory (RAM) of the computer.
- In general every programming language is containing three categories of data types. They are

1. Fundamental or primitive data types
2. Derived data types
3. User defined data types

Primitive data types

- **Primitive** data types are those whose variables allows us to store only one value but they never allows us to store multiple values of same type. This is a data type whose variable can hold maximum one value at a time.

Example

```
int a; // valid
a=10; // valid
 a=10, 20, 30; // invalid
```

- Here "a" store only one value at a time because it is primitive type variable.

Derived data types

- **Derived** data types are those whose variables allow us to store multiple values of same type. But they never allow to store multiple values of different types. These are the data type whose variable can hold more than one value of similar type. In general derived data type can be achieve using array.

Example

```
int a[] = {10,20,30}; // valid
int b[] = {100, 'A', "ABC"}; // invalid
```

- Here derived data type store only same type of data at a time not store integer, character and string at same time.

User defined data types

- User defined data types are those which are developed by programmers by making use of appropriate features of the language.
- User defined data types related variables allows us to store multiple values either of same type or different type or both. This is a data type whose variable can hold more than one value of dissimilar type, in java it is achieved using class concept.

Example

```
Student s = new Student();
```

In java we have eight data type which are organized in four groups. They are

- Integer category data types
- Character category data types
- Float category data types
- Boolean category data types

Integer category data types

- These category data types are used for storing integer data in the main memory of computer by allocating sufficient amount of memory space.
- Integer category data types are divided into four types which are given in following table

	Data Type	Size	Range
1.	**Byte**	**1**	**+ 127 to -128**
2.	**Short**	**2**	**+ 32767 to - 32768**
3.	**Int**	**4**	**+ x to - (x+1)**
4.	**Long**	**8**	**+ y to - (y+1)**

<u>Character category data types</u>

- A character is an identifier which is enclosed within single quotes. In java to represent character data, we use a data type called char. This data type takes two byte since it follows Unicode character set.

Data Type	Size (Byte)	Range
Char	2	232767- to -32768

Why Java take 2 byte of memory for store character?

- Java support more than 18 international languages so java take 2 byte for characters, because for 18 international language 1 byte of memory is not sufficient for storing all characters and symbols present in 18 languages. Java supports Unicode but c support ascii code. In ascii code only English language are present, so for storing all English latter and symbols 1 byte is sufficient. Unicode character set is one which contains all the characters which are available in 18 international languages and it contains 65536 characters

Float category data types

- Float category data type are used for representing float values. This category contains two data types, they are in the given table

Why Boolean data types take zero byte of memory?

- Boolean data type takes zero bytes of main memory space because Boolean data type of java implemented by Sun Micro System with a concept of flip - flop. A flip - flop is a general purpose register which stores one bit of information (one true and zero false).

Data Type	Default Value	Default size
Boolean	false	1 bit
Char	'\u0000'	2 byte
Byte	0	1 byte
Short	0	2 byte
Int	0	4 byte
Long	0L	8 byte
Float	0.0f	4 byte

Double	0.0d	8 byte

Example for default value of all primitive data type of Java.

```
class defaultdemo
{
static byte b;
static short s;
static int n;
static long l;
static float f;
static double d;
static char c;
static boolean bool;
public static void main(String[] args)
{
System.out.println("The default values of all primitive data types are:");
System.out.println("Byte :"+b);
System.out.println("Short :"+s);
System.out.println("Int :"+n);
System.out.println("Long :"+l);
System.out.println("Float :"+f);
System.out.println("Double :"+d);
System.out.println("Char :"+c);
System.out.println("Boolean :"+bool);
}
}
```

Output:

```
The default values of all primitive data types are:
Byte :0
Short :0
Int :0
Long :0
Float :0.0
Double :0.0
Char :
Boolean :false
```

Variable in Java

Variable is an identifier which holds data or another one variable is an identifier whose value can be changed at the execution time of program. Variable is an identifier which can be used to identify input data in a program.

Syntax:

```
Variable_name =  value;
```

Example:

```
a = 10;
```

Rules to declare a Variable

- Every variable name should start with either **alphabets** or **underscore** (_) or **dollar** ($) symbol.
- **No space** are allowed in the variable declarations.

- Except underscore (_) **no special** symbol are allowed in the **middle** of variable declaration
- Variable name always should exist in the **left hand** side of assignment operators.
- Maximum length of variable is **64 characters**.
- No keywords should access variable name.

Note: Actually a variable also can start with ¥, ¢, or any other currency sign.

Example of Variable Declaration

```
class Sum
{
 public static void main (String[] args)
  {
   Int   _a, ¢b, ¥c, $d, result;
   _a=10;
   ¢b=20;
   ¥c=30;
   $d=40;
   result=_a+¢b+¥c+$d;
   System.out.println("Sum :" +result);
  }
}
```

Output:

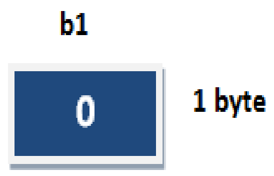
```
        Sum: 100
```

Variable declarations

- In which sufficient memory will be allocated and holds default values.

Syntax

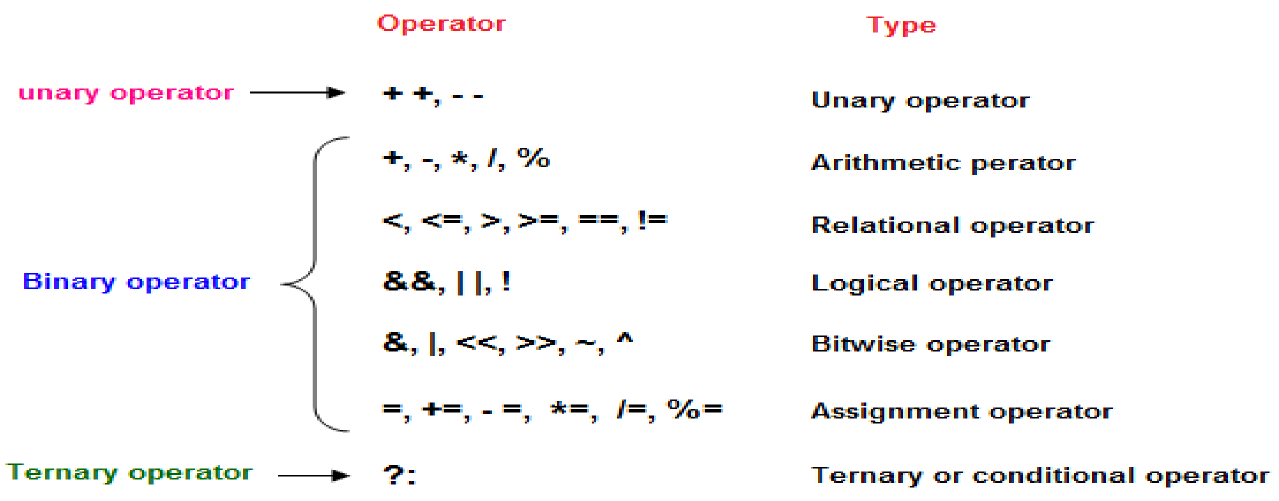
```
          Datatype  variable_name;
               byte b1;
```



<u>Operators in Java</u>

- **Operator** is a special symbol that tells the compiler to perform specific mathematical or logical Operation. Java supports following lists of operators.

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Ternary or Conditional Operators



	Operator	Type
unary operator	+ +, - -	Unary operator
Binary operator	+, -, *, /, %	Arithmetic perator
	<, <=, >, >=, ==, !=	Relational operator
	&&, | |, !	Logical operator
	&, |, <<, >>, ~, ^	Bitwise operator
	=, +=, - =, *=, /=, %=	Assignment operator
Ternary operator	?:	Ternary or conditional operator

OOP's Concept in Java

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects.

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Object

- Object is the physical as well as logical entity where as class is the only logical entity
- Object is a instance of class, object has state and behaviors.

An Object in java has three characteristics:

1. **State**
2. **Behavior**
3. **Identity**

1. **State:** Represents data (value) of an object.

2. **Behavior:** Represents the behavior (functionality) of an object such as deposit, withdraw etc.

3. **Identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user.

Class

- Class is a blue print which is containing only list of variables and method and no memory is allocated for them. A class is a group of objects that has common properties

Encapsulation

- Encapsulation is a process of wrapping of data and methods in a single unit is called encapsulation. The main advantage of using of encapsulation is to secure the data from other methods, when we make a data private then these data only use within the class, but these data not accessible outside the class.

Real life example of Encapsulation in Java

- The common example of encapsulation is capsule. In capsule all medicine are encapsulated in side capsule.

Benefits of encapsulation

- Provides abstraction between an object and its clients.
- Protects an object from unwanted access by clients.
- Example: A bank application forbids (restrict) a client to change an Account's balance.

Example:

```
class Employee
{
private String name;

public String getName()
{
return name;
}
public void setName(String name){
this.name=name;
}
}
```

```
class Demo
{
public static void main(String[] args)
{
Employee e=new Employee();
e.setName("Rohit");
System.out.println(e.getName());
}
}
```

Output:

```
Rohit
```

Abstraction

- Abstraction is the concept of exposing only the required essential characteristics and behavior with respect to a context.
- Hiding of data is known as data abstraction. In object oriented programming language this is implemented automatically while writing the code in the form of class and object.

Example of Abstraction

```
class Customer

{

int account_no;

float balance_Amt;

String name;

int age;

String address;

void balance_inquiry()

{

/* to perform balance inquiry only account number

is required that means remaining properties

are hidden for balance inquiry method */
}
void fund_Transfer()
{
/* To transfer the fund account number and
balance is required and remaining properties
are hidden for fund transfer method */
}
```

Inheritance in Java :

- The process of obtaining the data members and methods from one class to another class is known as inheritance
- In the inheritance the class which is give data members and methods is known as base or super or parent class.
- The class which is taking the data members and methods is known as sub or derived or child class.
- The data members and methods of a class are known as features.
- The concept of inheritance is also known as re-usability or extendable classes or sub classing or

Use Inheritance

- For Method Overriding (used for Runtime Polymorphism).
- It's main uses are to enable polymorphism and to be able to reuse code for different classes by putting it in a common super class
- For code Re-usabilityderivation.

Syntax of Inheritance

```
class Subclass-Name extends Superclass-Name
{
  //methods and fields
}
```

Advantage of inheritance

- Application development time is less.
- Application take less memory.
- Application execution time is less.
- Application performance is enhance (improved).
- Redundancy (repetition) of the code is reduced or minimized so that we get consistence results and less storage cost.

```
class Employee
{
float salary=50000;
}
class Programmer extends Employee
{
float bonous=10000;
public static void main(String args[])
{
Employee obj=new Employee ();
System.out.println("Salary :"+obj.salary);
System.out.println("Bonous :"+obj.bonous);
}
}
```

Output:

```
Salary: 50000
```

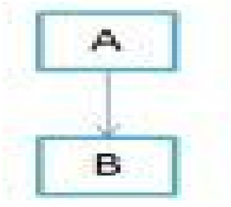
Bonous: 10000

Types of Inheritance

1. Single inheritance
2. Multilevel inheritance
3. Hierarchical inheritance
4. Multiple inheritance
5. Hybrid inheritance

1. Single inheritance:

- When a class extends another one class only then we call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a parent class of B and B would be a child class of A.



(a) Single Inheritance

Syntax:

```
class A{
        //Do something
}
class B extends A{
        //Do something
}
```

Example of Single Inheritance:

```
Class A
{
  public void methodA()
  {
    System.out.println("Base class method");
  }
}

Class B extends A
{
  public void methodB()
  {
    System.out.println("Child class method");
  }
  public static void main(String args[])
  {
    B obj = new B();
    obj.methodA(); //calling super class method
    obj.methodB(); //calling local method
 }
}
```

Another Example of Single Inheritance:

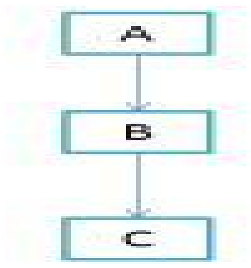
```
class Employee {
   void salary() {
      System.out.println("Salary= 500000");
   }
}

class Programmer extends Employee {
// Programmer class inherits from Employee class
   void bonus() {
      System.out.println("Bonus=10000");
   }
}

class single_inheritance {
   public static void main(String args[]) {
      Programmer p = new Programmer();
      p.salary(); // calls method of super class
      p.bonus(); // calls method of sub class
   }
}
```

Output:

```
Salary= 500000
Bonus=10000
```

2. Multilevel inheritance

- Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A



(d) Multilevel Inheritance

Example:

```
class A
{
  public void methodA()
  {
    System.out.println("Class A method");
  }
}
class B extends A
{
public void methodB()
{
System.out.println("class B method");
}
}
class C extends B
{
  public void methodC()
  {
    System.out.println("class C method");
  }
  public static void main(String args[])
  {
    C obj = new C();
    obj.methodA(); //calling grand parent class method
    obj.methodB(); //calling parent class method
    obj.methodC(); //calling local method
 }
}
```

Output:

```
Class A method
Class B method
Class C method
```

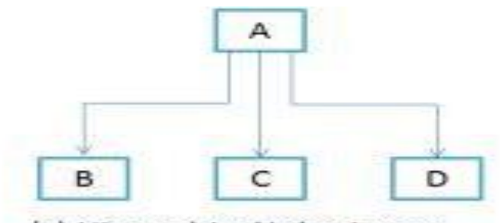
Another Example:

```
class Faculty
{
float total_sal=0, salary=30000;
}

class HRA extends Faculty
{
float hra=3000;
}

class DA extends HRA
{
float da=2000;
}

class Science extends DA
{
float bonous=2000;
public static void main(String args[])
{
Science obj=new Science();
obj.total_sal=obj.salary+obj.hra+obj.da+obj.bonous;
System.out.println("Total Salary is:"+obj.total_sal);
}
}
```

Total Salary is: 57000.0

3. Hierarchical Inheritance:

- In such kind of inheritance one class is inherited by many sub classes. In below example class B, C and D inherits the same class A. A is parent class (or base class) of B, C & D



(c) Hierarchical Inheritance

Example:

```
class A
{
  public void methodA()
  {
    System.out.println("method of Class A");
  }
}
class B extends A
```

```
{
   public void methodB()
   {
      System.out.println("method of Class B");
   }
}
class C extends A
{
  public void methodC()
  {
     System.out.println("method of Class C");
  }
}
class D extends A
{
  public void methodD()
  {
     System.out.println("method of Class D");
  }
}
class JavaExample
{
  public static void main(String args[])
  {
     B obj1 = new B();
     C obj2 = new C();
     D obj3 = new D();
     //All classes can access the method of class A
     obj1.methodA();
```

```
    obj2.methodA();
    obj3.methodA();
  }
}
```

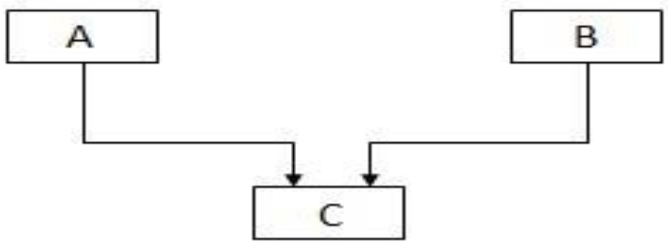
Output:

```
method of Class A
method of Class A
method of Class A
```

Another Example:

4. Multiple Inheritance

- Multiple inheritance is inheriting properties of two or more parent classes to one child class.
- As given in the below diagram class A and class B is being inherited by the child class C.



Example:

```
//first parent class
class ParentA
{
   //same signature method in both class named as walk()
          void msg()
          {
          System.out.println(“Method of Parent A”);
          }
}

//second parent class
class ParentB
{
          void msg()
          {
          System.out.println(“Method of Parent B");
          }
}

//multiple inheritance in child class results in error
class childC extends ParentA, ParentB
{
          public static void main(String[] args)
          {
                    childC object = new childC();
                    object.msg();
}
}
```

Output:

Compile time Error

- In the above code the compiler will get confuse which method to call when object of child class calls msg() method and results in compile time error.
- **Hence,** Java does not support multiple inheritance because it can lead to increased complexity and ambiguity
- The concept of multiple inheritance is not supported in java through concept of classes but it can be supported through the concept of interface.

Syntax of Multiple Inheritance Using Interface

```
public interface A{
        //Do Something
}
public interface B extends A{
        //Do Something
}
public interface C extends A{
        //Do Something
}
```

Example of Multiple Inheritance Using Interface

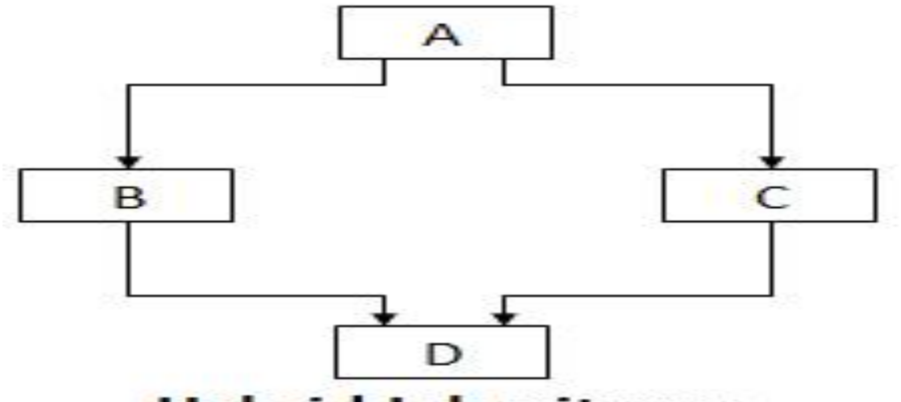
```
interface vehicleone{
        int  speed=90;
        public void distance();
}

interface vehicletwo{
        int distance=100;
        public void speed();
}

class Vehicle  implements vehicleone,vehicletwo{
        public void distance(){
                int  distance=speed*100;
                System.out.println("Distance travelled is: "+distance);
        }
        public void speed(){
                int speed=distance/100;
        }
}

class MultipleInheritanceUsingInterface{
        public static void main(String args[]){
                System.out.println("Vehicle");
                obj.distance();
                obj.speed();
        }
}
```

Output

```
Distance travelled is: 9000
```

5. Hybrid inheritance:

- **Combination of any inheritance type**
- **In the combination if one of the combination is multiple inheritance then the inherited combination is not supported by java through the classes concept but it can be supported through the concept of interface**



Hybrid Inheritance

Example of Hybrid Inheritance

```
public class A
{
   public void dispA()
   {
      System.out.println("method of ClassA");
   }
}
public class B extends A
{
   public void show()
   {
      System.out.println("method of ClassB");
   }
   public void dispB()
   {
      System.out.println("disp() method of Class B");
   }
}
public class C extends A
{
   public void show()
   {
      System.out.println("method of Class C");
   }
   public void dispC()
   {
      System.out.println("method of Class C");
   }
}
public class D extends B,C // Compiler Error
{
   public void dispD()
   {
      System.out.println("method of ClassD");
   }
   public static void main(String args[])
   {
        D d = new D();
        d.dispD();
        d.show(); //Confusion happens here which show method to call
   }
}
```

Output

Compiler Error

- We all know that we cannot extend Class B and C to D at the same time but the point to be noted here is why it is not allowed. Since we have same show() method in both Class B and D compiler will not be able to differentiate which method to call whereas if we are using interface we can avoid that ambiguity.

Implementation of Hybrid Inheritance with Interfaces:

```
class A
{
    public void dispA()
    {
        System.out.println("method of Class A");
    }
}
interface B
{
    public void show();
}
interface C
{
    public void show();
}
 class D extends A implements B,C
{
    public void show()
    {
        System.out.println("method implementation");
    }
    public void dispD()
    {
        System.out.println("method of Class D");
    }
    public static void main(String args[])
    {
        D d = new D();
        d.dispD();
        d.show();
    }
}
```

Output :

```
disp() method of Class D
show() method implementation
```

Interface in Java

- Interface is similar to class which is collection of public static final variables (constants) and abstract methods.

- The interface is a mechanism to achieve fully abstraction in java. There can be only abstract methods in the interface. It is used to achieve fully abstraction and multiple inheritance in Java.

Use Interface

- It is used to achieve fully abstraction.
- By using Interface, you can achieve multiple inheritance in java.
- It can be used to achieve loose coupling.

Syntax

```
interface ClassName
{
 datatype variablename=value;
 //Any number of final, static fields
 returntype methodname(list of parameters or no parameters)
 //Any number of abstract method declarations
}
```

Example of Interface in Java

```
interface Person
{
void run();  // abstract method
}
class Test implements Person
{
public void run()
{
System.out.println("Run fast");
}
public static void main(String args[])
 {
 Test obj = new Test();
 obj.run();
 }
}
```

Output:

```
Run fast
```

<u>Implementing of Interfaces:</u>

- A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

Example Interface in Java

```
interface Person
        {
                void run();
                }
                        class Employee implements Person
                        {
                                public void run()
                                {
                                        System.out.println("Run fast");
                                        }
                                                public static void main(String args[])
                                        {
                                                 Employee obj = new Employee ();
                                                 obj.run();
                                        }
                                }

                        }
```

<u>Output:</u>

```
                Run fast
```

Package in Java

A package is a collection of similar types of classes, interfaces and sub-packages.

Advantage of package

- Package is used to categorize the classes and interfaces so that they can be easily maintained
- Application development time is less, because reuse the code
- Application memory space is less (main memory)
- Application execution time is less
- Application performance is enhance (improve)
- Redundancy (repetition) of code is minimized
- Package provides access protection.
- Package removes naming collision.

Type of package

1. Predefined or built-in package
2. User defined package

1. Predefined or built-in package

- These are the package which are already designed by the Sun Microsystem and supply as a part of java API, every predefined package is collection of predefined classes, interfaces and sub-package.

2. User defined package

- If any package is design by the user is known as user defined package. User defined package are those which are developed by java programmer and supply as a part of their project to deal with common requirement.

Rules to create user defined package

- package statement should be the first statement of any package program.
- Choose an appropriate class name or interface name and whose modifier must be public.
- Any package program can contain only one public class or only one public interface but it can contain any number of normal classes.
- Package program should not contain any main class (that means it should not contain any main())
- modifier of constructor of the class which is present in the package must be public. (This is not applicable in case of interface because interface have no constructor.)
- The modifier of method of class or interface which is present in the package must be public (This rule is optional in case of interface because interface methods by default public)
- Every package program should be save either with public class name or public Interface name

Example of package program

```
package pack;
public class A
{
public void show()
{
System.out.println("Example of Package");
}
}


/* Package program which is save with A.java and compile by javac -d . A.java */
```

Method Overloading in Java

- Whenever same method name is exiting multiple times in the same class with different number of parameter or different order of parameters or different types of parameters is known as method overloading.

Advantage of Method Overloading in Java

- One significant advantage of method overriding is that a class can give its specific execution to an inherited method without having the modification in the parent class (base class).

Syntax

```
class  class_Name
{
        Returntype  method()
        {.........}
        Returntype  method(datatype1 variable1)
        {.........}
        Returntype  method(datatype1 variable1, datatype2 variable2)
        {.........}
        Returntype  method(datatype2 variable2)
        {.........}
        Returntype  method(datatype2 variable2, datatype1 variable1)
        {.........}

}
```

Example Method Overloading in Java

```
class Addition
{
void sum(int a, int b)
{
System.out.println(a+b);
}
void sum(int a, int b, int c)
{
```

```
System.out.println(a+b+c);
}
public static void main(String args[])
{
Addition obj=new Addition();
obj.sum(10, 20);
obj.sum(30, 40, 50);
}
}
```

Output:

```
30
120
```

<u>Method Overriding in Java</u>

- Whenever same method name is existing in both base class and derived class with same types of parameters or same order of parameters is known as method Overriding. Here we will discuss about Overriding in Java.

<u>Advantage of Java Method Overriding</u>

- Method Overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method Overriding is used for Runtime Polymorphism

<u>Rules for Method Overriding</u>

- Method must have same name as in the parent class.
- Method must have same parameter as in the parent class.
- Must be IS-A relationship (inheritance).
- The data types of the arguments along with their sequence must have to be preserved as it is in the overriding method.
- Any method which is overriding can throw any unchecked exceptions, in spite of whether the overridden method usually method of parent class might throw an exception or not.
- The private, static and final methods can't be overridden as they are local to the class

Example Method Overriding in Java

```
class Walking
{
void walk()
{
System.out.println("Man walking fastly");
}
}
class Man extends walking
{
void walk()
{
System.out.println("Man walking slowly");
}
}

class OverridingDemo
{
public static void main(String args[])
{
Walking obj = new Walking();
obj.walk();
}
}
```

Output

```
Man walking slowly
```

Abstract class in Java

- We know that every Java program must start with a concept of class that is without the class concept there is no Java program perfect.

In Java programming we have two types of classes they are

1. Concrete class
2. Abstract class

1. <u>Concrete class in Java</u>

- A concrete class is one which is containing fully defined methods or implemented method.

<u>Example</u>

```
class  Msg
{
void  display()
{
System.out.println("Good Morning");
}
}
```

Here, Msg class is containing a defined method and object can be created directly.

<u>Create an object</u>

```
Msg obj=new  Msg();
obj.display();
```

Every concrete class has specific features and these classes are used for specific requirement, but not for common requirement.

<u>If we use concrete classes for fulfill common requirements than such application will get the following limitations.</u>

- Application will take more amount of memory space (main memory).
- Application execution time is more.
- Application performance is decreased.

2. <u>Abstract class in Java</u>

- A class that is declared with abstract keyword, is known as abstract class. An abstract class is one which is containing some defined method and some undefined method. In java programming undefined methods are known as un-Implemented, or abstract method.

Syntax

```
abstract class className
{
......
}
```

Example

```
abstract class A
{
.....
}
```

- If any class have any abstract method then that class become an abstract class.

Example

```
class Vachile
{
abstract void Bike();
}
```

- Class Vachile is become an abstract class because it have abstract Bike() method.

Make class as abstract class

- To make the class as abstract class, whose definition must be preceded by a abstract keyword.

Example

```
abstract class Vachile
{
............
………..
}
```

Abstract method:

- An abstract method is one which contains only declaration or prototype but it never contains body or definition. In order to make any undefined method as abstract whose declaration is must be predefined by abstract keyword.

Syntax:

```
abstract ReturnType methodName(List of formal parameter)
```

Example:

```
abstract   void  sum();
abstract  void  diff(int, int);
```

Example of abstract class

```
abstract class Vachile {
 abstract void speed();  // abstract method
}
class Bike extends Vachile {
void speed() {
System.out.println("Speed limit is 40 km/hr");
}
public static void main(String args[])
{
 Vachile obj = new Bike(); //indirect object creation
 obj.speed();
 }
}
```

Output

```
Speed limit is 40 km/hr
```