**Inheritance:**

* Inheritance is an important pillar of OOP(Object-Oriented Programming).
* It is the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class.
* Inherited is achieved by **extend** keyword
* Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship

**Real time example:**

After Dhirubhai ambani died,his sons mukesh and anil ambani have inherited their father wealth,mukesh extended his wealth with the help of his father wealth

**Importance Terminology:**

**Super Class:** The class whose features are inherited is known as superclass(or a **base class** or a **parent class**).

Example Dhirubhai ambani

**Sub Class:** The class that inherits the other class is known as a subclass(or a derived class, extended class, or child class).

The subclass can add its own fields and methods in addition to the superclass fields and methods.

**Reusability:** Inheritance supports the concept of **“reusability**”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

**Syntax :**

class derived-class extends base-class { //methods and fields }

## 1.Single Inheritance Example

When a class inherits another class, it is known as a *single inheritance*.

class Animal{

public void eat(){

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

}

class Dog extends Animal{

Public void bark(){

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

}

class TestInheritance{

public static void main(String args[]){

Dog d=new Dog();

d.bark();

d.eat();

}}

## 2.Multilevel Inheritance Example

When there is a chain of inheritance, it is known as *multilevel inheritance*.



**class** Animal{

**void** eat(){System.out.println("eating...");}

}

**class** Dog **extends** Animal{

**void** bark(){System.out.println("barking...");}

}

**class** BabyDog **extends** Dog{

**void** weep(){System.out.println("weeping...");}

}

**class** TestInheritance2{

**public** **static** **void** main(String args[]){

BabyDog d=**new** BabyDog();

d.weep();

d.bark();

d.eat();

}}

## 3.Hierarchical Inheritance Example

When two or more classes inherits a single class, it is known as *hierarchical inheritance*

**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();//

## **Q) Why multiple inheritance is not supported in java?**

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

**class** A{

**void** msg(){System.out.println("Hello");}

}

**class** B{

**void** msg(){System.out.println("Welcome");}

}

**class** C **extends** A,B{//suppose if it were

**public** **static** **void** main(String args[]){

   C obj=**new** C();

   obj.msg();//Now which msg() method would be invoked?

}

}

## 2.Java Encapsulation

# Encapsulation in Java

**Encapsulation in Java** is a *process of binding code and data together into a single unit*,

for example, a capsule which is mixed of several medicines.

### Advantage of Encapsulation in Java

* It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.
* By providing only a setter or getter method, you can make the class **read-only or write-only**.
* It provides you the **control over the data**. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.
* The encapsulate class is **easy to test**. So, it is better for unit testing.

**Package:**

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

## How to access package from another package?

There are three ways to access the package from outside the package.

1. import package.\*;
2. import package.classname;
3. fully qualified name.

# Access Modifiers in Java

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

**There are four types of Java access modifiers:**

**Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

**Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

**Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

**Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

### Understanding Java Access Modifiers

Let's understand the access modifiers in Java by a simple table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **within class** | **within package** | **outside package by subclass only** | **outside package** |
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

**3.Polymorphism**

* Basically poly means many morph mean different forms - Many different forms
* Polymorphism in Java is a concept of performing a single action in many different forms
* Polymorphism is the capability of a method to do different things based on the object

Example We act differently to different persons like To Manager I am Lead,To parents I am child etc

**Types of Polymorphism:**

1.Static/CompileTime/early binding/**Method overloading**

* If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

## Advantage of method overloading

Method overloading *increases the readability of the program*

### Different ways to overload the method

There are two ways to overload the method in java

By changing number of arguments

**class** Adder{

**static** **int** add(**int** a,**int** b){**return** a+b;}

**static** **int** add(**int** a,**int** b,**int** c){**return** a+b+c;}

}

**class** TestOverloading1{

**public** **static** **void** main(String[] args){

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(11,11,11));

}}

By changing the data type

**class** Adder{

**static** **int** add(**int** a, **int** b){**return** a+b;}

**static** **double** add(**double** a, **double** b){**return** a+b;}

}

**class** TestOverloading2{

**public** **static** **void** main(String[] args){

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(12.3,12.6));

}}

2.Dynamic/RunTime/Late binding / Methodoverriding

Methodoverriding

* If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

**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

Example -

//Java Program to illustrate the use of Java Method Overriding

//Creating a parent class.

**class** Vehicle{

//defining a method

**void** run(){System.out.println("Vehicle is running");}

}

//Creating a child class

**class** Bike2 **extends** Vehicle{

//defining the same method as in the parent class

**void** run(){System.out.println("Bike is running safely");}

**public** **static** **void** main(String args[]){

Bike2 obj = **new** Bike2();//creating object

obj.run();//calling method

}

}

### 3.Abstraction

* General english term **abstraction means it is the process of working with ideas** rather than its implementation Abstraction means Unimplemented ideas
* When ideas implemented ,Abstraction percentage is 0
* When ideas NOT implemented,Abstraction percentage is 100
* Example – If your idea is to build a 10 storey building,If you build 6 floors then the abstraction is 40 percentage

If you build 10 floors,then your abstraction is 0

**Abstraction in Java:**

* **Hide the implementation details** and **showing only functionality to the user.**
* Abstraction means **Unimplemented ideas**

**for example**, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

* Abstract class must be declared with abstract keyword
* Abstract class can have abstract and non-abstract method

**Simply Abstraction in Java means Unimplemented**

**There are two ways to achieve abstraction in java**

* Abstract class (0 to 100%)
* Interface (100%)

Percentage of abstraction is calculated based on how much implementation details we know

Example Constructing a building of 4 floors,if we don’t know what are all things needed on 3rd and 4th floor then it **50 percentage abstraction**

#### Points to Remember

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.(Meaning we cannot create object for Abstract method)
* It can have [constructors](https://www.javatpoint.com/java-constructor) and static methods also.
* It can have final methods which will force the subclass not to change the body of the method.

### Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

**Example of abstract method**

**abstract** **void** printStatus();//no method body and abstract

### Example of Abstract class that has an abstract method

In this example, Bike is an abstract class that contains only one abstract method run. Its implementation is provided by the Honda class.

**abstract** **class** Bike{

**abstract** **void** run();

}

**class** Honda4 **extends** Bike{

**void** run(){

System.out.println("running safely");}

**public** **static** **void** main(String args[]){

Bike obj = **new** Honda4();

obj.run();

}

}

**Interfaces in java:**

* It is used to provide **total abstraction**
* It is used to hiding the implementation from user
* Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).
* If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract.
* To use Interface you have to use keyword Interface
* To implement an Interface Use Implement keyword
* Variable in interface are public, static and final

**Why do we use interface ?**

* It is used to achieve 100 percentage total abstraction.(hiding the implementation from user)
* Since java does not support multiple inheritance , but by using interface we can achieve multiple inheritance .
* Interfaces are used to implement abstraction. So the question arises why use interfaces when we have abstract classes? The reason is, abstract classes may contain non-final variables, whereas variables in interface are public, static and final

**How Interface achieves 100 percentage total abstraction**

* The user who want to **use the methods of the interface**, he only knows the classes that implement this interface and their methods, information about the implementation is completely hidden from the user, thus achieving 100% abstraction

**Why Java NOT supported Multiple Inheritance?**

Example – If there are classses like Class A ,Class B and Class C

Class C extends Class A,Class B // this is called multiple inheritance

Class A {

Walk() // method

}

Class B {

Walk() // method

}

Clas C {

Main method

Object creation

Trited ot call Obj.walk(); // java is confused which method to call whether class Awalk or class B walk ,so it causes ambuiguity this is the reason multiple inheritance not supported in java

**How Interface going to solve this Multiple Inheritance problem?**

it is because its implementation is provided by the implementation class.

**Example** -

Class A is a abstract class

have abstract method called swim();

Class B extends Class A

{

Swim()

{

Syso ( “swim in beach”) // here implentation is done

}

Class C extends Class A ,Class B

Here Class B method is called bcz implementation done here

**In a real scenario,** an interface is **defined by someone else**, but its **implementation is provided by different implementation providers**. Moreover, **it is used by someone else**. The implementation part is hidden by the user who uses the interface.

**The relationship between classes and interfaces**



**How to declare an interface?**

An interface is declared by using the interface keyword.

It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.

A class that implements an interface must implement all the methods declared in the interface.

**Syntax:**

interface <interface\_name>{

// declare constant fields

// declare methods that abstract

// by default.

}

## Java Interface Example

In this example, the Printable interface has only one method, and its implementation is provided in the A6 class.

**interface** printable{

**void** print();

}

**class** A6 **implements** printable{

**public** **void** print(){System.out.println("Hello");}

**public** **static** **void** main(String args[]){

A6 obj = **new** A6();

obj.print();

}

}

**What is the difference between abstract class and interface?**

* An abstract class permits y**ou to make functionality that subclasses can implement or override** whereas
* an interface only permits **you to state functionality but not to implement** it. A class can extend only one abstract class while a class can implement multiple interfaces

**Interface**

* To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class with the implements keyword (instead of extends). The body of the interface method is provided by the "implement" class

// Interface  
interface Animal {  
 public void animalSound(); // interface method (does not have a body

)  
 }

// Dog "implements" the Animal interface

class Dog implements Animal {  
 public void animalSound() {  
 // The body of animalSound() is provided here  
 System.out.println("The pig says: wee wee");  
 }  
}  
}  
  
class Main {  
 public static void main(String[] args) {  
 Dog myDog= new Dog(); // Create a Pig object  
 myDog.animalSound();  
  
 }  
}

**Notes on Interfaces:**

* Like abstract classes, interfaces cannot be used to create objects (in the example above, it is not possible to create an "Animal" object in the MyMainClass)
* Interface methods do not have a body - the body is provided by the "implement" class
* On implementation of an interface, you must override all of its methods
* Interface methods are by default abstract and public
* Interface attributes are by default public, static and final
* An interface cannot contain a constructor (as it cannot be used to create objects)