# Inheritance and Interface

## Inheritance

- Inheritance is one of the key features of Object Oriented Programming.
- Inheritance provided mechanism that allowed **a class to inherit property of another class**.
- When a Class extends another class it inherits all non-private members including fields and methods.
- Inheritance in Java can be best understood in terms of Parent and Child relationship, also known as **Super class**(Parent) and **Sub class**(child) in Java language.
- Class uses **extends** keyword to inherit properties of another class.

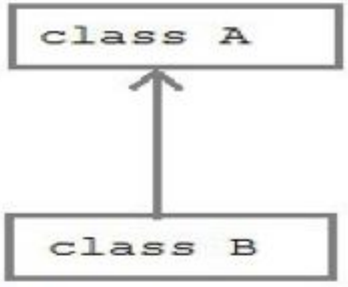
## Simple example of Inheritance

```
class Parent
{
  public void p1()
{
    System.out.println("Parent method");
  }
}
public class Child extends Parent { public void c1()
  {
    System.out.println("Child method");
  }
  public static void main(String[] args)
  {
    Child cobj = new Child();
    cobj.c1();   //method of Child class
    cobj.p1();   //method of Parent class
  }
}
```

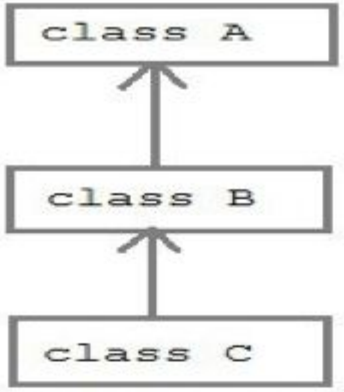
## Types of Inheritance

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance

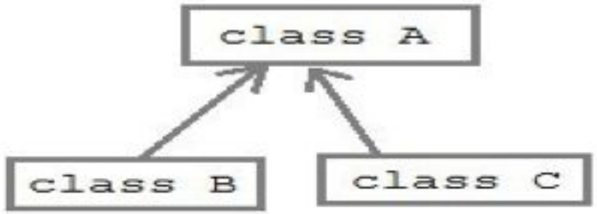
**NOTE :**Multiple inheritance is not supported in java



Simple Inheritance



Multilevel inheritance



Heirarchical inheritance

## Why multiple inheritance is not supported in Java

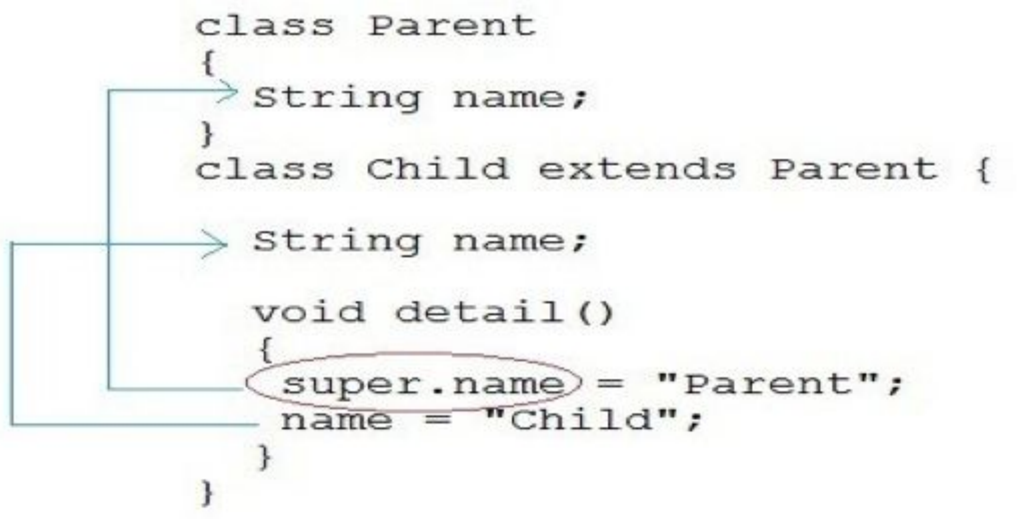
- To remove ambiguity.
- To provide more maintainable and clear design.

## super keyword

The **`super`** keyword refers to superclass (parent) objects.

It is used to call superclass methods, and to access the superclass constructor.

The most common use of the **`super`** keyword is to eliminate the confusion between superclasses and subclasses that have methods with the same name.



```
class Parent
{
  String name;
}
class Child extends Parent {

  String name;

  void detail()
  {
   super.name = "Parent";
   name = "Child";
  }
}
```

## Polymorphism

**Polymorphism in Java** is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

## Types of Polymorphism

**Compile-Time Polymorphism:**

- Achieved through method overloading.
- Resolved during compile-time.
- Provides faster execution as method calls are resolved at compile time.
- Example: Multiple methods with the same name but different signatures in a class.

**Dynamic Polymorphism:**

- Achieved through method overriding.
- Resolved during runtime.
- Provides flexibility and extensibility as it allows a single interface to be used for different data types.
- Example: A method in a superclass is overridden by a method in a subclass.

## Method Overloading

- Method overloading in Java is a feature that allows a class to have more than one method with the same name, but with different parameter lists (i.e., different type, number, or both of parameters).
- Method overloading increases the readability of the program and is an example of compile-time polymorphism.

```
Class Add
{
Void sum(int a,int b)
{
System.out.println("Addition":(a+b));
}
Void sum(int a,int b,int c)
{
System.out.println("Addition":(a+b+c));
}
Public static void main(String argos[])
{
Add a=new Add();
a.sum(10,20);
a.sum(11,22,33);}}
```

## Method Overriding

- Method overriding in Java is a feature that allows a subclass to provide a specific implementation for a method that is already defined in its superclass.
- The method in the subclass should have the same name, return type, and parameters as the method in the superclass.
- Method overriding is used to achieve runtime polymorphism and to implement specific behavior in the subclass.

```
class Parent
{
 public void show()
  {
  System.out.println("Hello from Parent");
  }
}
class Child extends Parent
{
 public void show()
  {
  System.out.println("Hello from Child");
  }

}
```

| Method Overloading | Method Overriding |
|---|---|
| Method overloading is a compile-time polymorphism. | Method overriding is a run-time polymorphism. |
| Method overloading helps to increase the readability of the program. | Method overriding is used to grant the specific implementation of the method which is already provided by its parent class or superclass. |
| It occurs within the class. | It is performed in two classes with inheritance relationships. |
| Method overloading may or may not require inheritance. | Method overriding always needs inheritance. |
| In method overloading, methods must have the same name and different signatures. | In method overriding, methods must have the same name and same signature. |

| | |
|---|---|
| In method overloading, the return type can or can not be the same, but we just have to change the parameter. | In method overriding, the return type must be the same or co-variant. |
| Static binding is being used for overloaded methods. | Dynamic binding is being used for overriding methods. |
| Poor Performance due to compile time polymorphism. | It gives better performance. The reason behind this is that the binding of overridden methods is being done at runtime. |
| Private and final methods can be overloaded. | Private and final methods can’t be overridden. |
| The argument list should be different while doing method overloading. | The argument list should be the same in method overriding. |

## Final Keyword

The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

- variable
- method
- class

- If you make any variable as final, you cannot change the value of final variable(It will be constant).
- If you make any method as final, you cannot override it.
- If you make any class as final, you cannot extend it.

## Abstract class

- If a class contain any abstract method then the class is declared as abstract class.
- An abstract class is never instantiated. It is used to provide abstraction. Although it does not provide 100% abstraction because it can also have concrete method.
- Syntax :

  abstract class class_name { }

  **Abstract method**

- Method that are declared without any body within an abstract class is known as abstract method.
- The method body will be defined by its subclass. Abstract method can never be final and static.
- Any class that extends an abstract class must implement all the abstract methods declared by the super class.
- Syntax :
- abstract return_type function_name ();

```
abstract class A

{

abstract void msg();

}

class B extends A

{

void msg()

{

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

}

public static void main(String[] args)

{

B b=new B();

b.msg();

}}
```

## Points to Remember

1. An abstract class must have an abstract method.
2. Abstract classes can have Constructors, Member variables and Normal methods.
3. Abstract classes are never instantiated.
4. When you extend Abstract class with abstract method, you must define the abstract method in the child class, or make the child class abstract.

## Interface

- Interface is a pure abstract class.
- Interface contains only abstract methods and constants.
- They are syntactically similar to classes, but you cannot create instance of an Interface and their methods are declared without any body.
- Interface is used to achieve complete abstraction in Java.
- When you create an interface it defines what a class can do without saying anything about how the class will do it.
- Syntax :

```
interface interface_name { }
```

## Rules for using Interface

- Methods inside Interface must not be static, final, native or strictfp.
- All variables declared inside interface are implicitly public static final

  variables(constants).
- All methods declared inside Java Interfaces are implicitly public and abstract, even if you don't use public or abstract keyword.
- Interface can extend one or more other interface.
- Interface cannot implement a class.
- Interface can be nested inside another interface.

```
interface Moveable

{

int AVG-SPEED = 40;

void move();

}

class Vehicle implements Moveable

{

public void move()

{

System .out. print in ("Average speed is"+AVG-SPEED");

}

public static void main (String[] arg)

{

Vehicle vc = new Vehicle();

vc.move();
```

## Difference between an interface and an abstract class?

| Abstract class | Interface |
|---|---|
| Abstract class is a class which contain one or more abstract methods, which has to be implemented by its sub classes. | Interface is a Java Object containing method declaration but no implementation. The classes which implement the Interfaces must provide the method definition for all the methods. |
| Abstract class is a Class prefix with an abstract keyword followed by Class definition. | Interface is a pure abstract class which starts with interface keyword. |
| Abstract class can also contain concrete methods. | Whereas, Interface contains all abstract methods and final variable declarations. |
| Abstract classes are useful in a situation that Some general methods should be implemented and specialization behavior should be implemented by child classes. | Interfaces are useful in a situation that all properties should be implemented. |

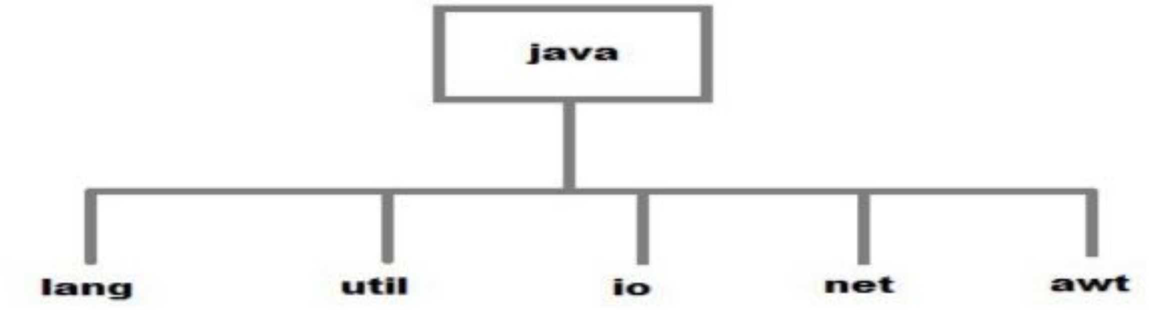
## Package

Package are used in Java, in-order to avoid name conflicts and to control access of class, interface and enumeration etc. A package can be defined as a group of similar types of classes, interface, enumeration and sub-package. Using package it becomes easier to locate the related classes.

**Package are categorized into two forms**

**Built-in Package:**-Existing Java package for example java.lang, java.util etc.

**User-defined-package:**- Java package created by user to categorized classes and interface



## Creating a user-defined package

Creating a package in java is quite easy. Simply include a package command followed by name of the package as the first statement in java source file.

```
package mypack;
public class employee
{
...statement;
}
```

```
package mypack;

class Book

{

String bookname;

String author;

Book(String b, String c)

{

this.bookname = b;

this.author = c;


}
public void show()

{

System.out.println(bookname+" "+ author);

}

}

class test

{

public static void main(String[] args)

{

Book bk = new Book("java","Herbert");

bk.show();

}

}
```