Inheritance

- Inheritance is the feature of OOPs ,which provides the facility to acquire one class properties such as instance variables and methods by another class.
- Every class by default inherits the fields and method of **Object class.**
- used to build new classes from existing classes.
- It builds IS-A relationship among classes.

Advantages:

- Reusability
- clear code structure
- Maintain abstraction
- Extensibility

Syntax:

```
class SuperClass
{
 // some code here
}
class SubClass extends SuperClass
{
 // some code here
}
```

- **The parent class is known as super class**
- **The child class is known as sub class**

Real-life example

For example, the class Student inherits from the class Person. Then although Student is a person, the reverse is not true. A Person need not be a Student. The class Student has properties that it does not share with class Person.

For instance, the Student has a marks-percentage, but the Person does not have.

```
class person

{

}

class student extends person

{

}
```

Important Terms in Java Inheritance

1. **Class**: Class is a user-defined datatype in Java that is basically a group of objects. It is a blueprint or template from which we create objects.

2. **Super Class**: The class whose features and functionalities are being inherited or used is known as the superclass or a base class or a parent class.

3. **Sub Class**: The class that inherits the properties and features from another class is known as a subclass or a derived class or extended class or child class. The subclass can add its own features and functions in addition to the fields and methods of its superclass or the parent class.

4. **The extends keyword**: The keyword extends is used by child class while inheriting the parent class.

5. **The super keyword**: The super keyword is similar to this keyword. It is a reference variable which holds the address of its super class.

Private Members in a Superclass

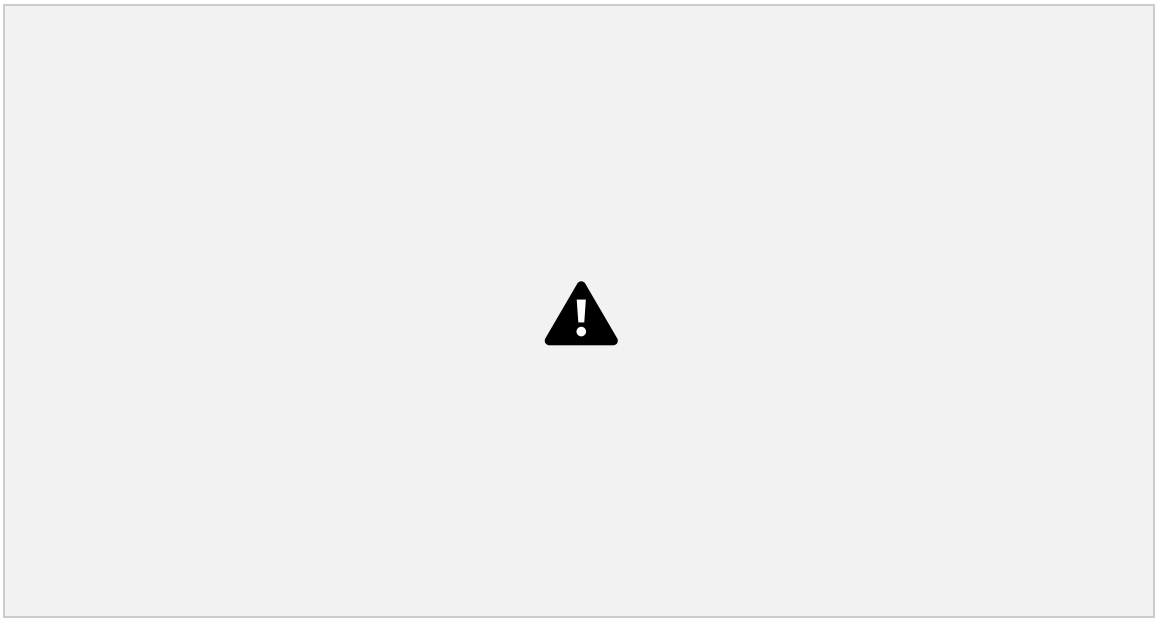
- A subclass does not inherit the private members of its parent class. However, if the superclass has public or protected methods for accessing its private fields, these can also be used by the subclass.

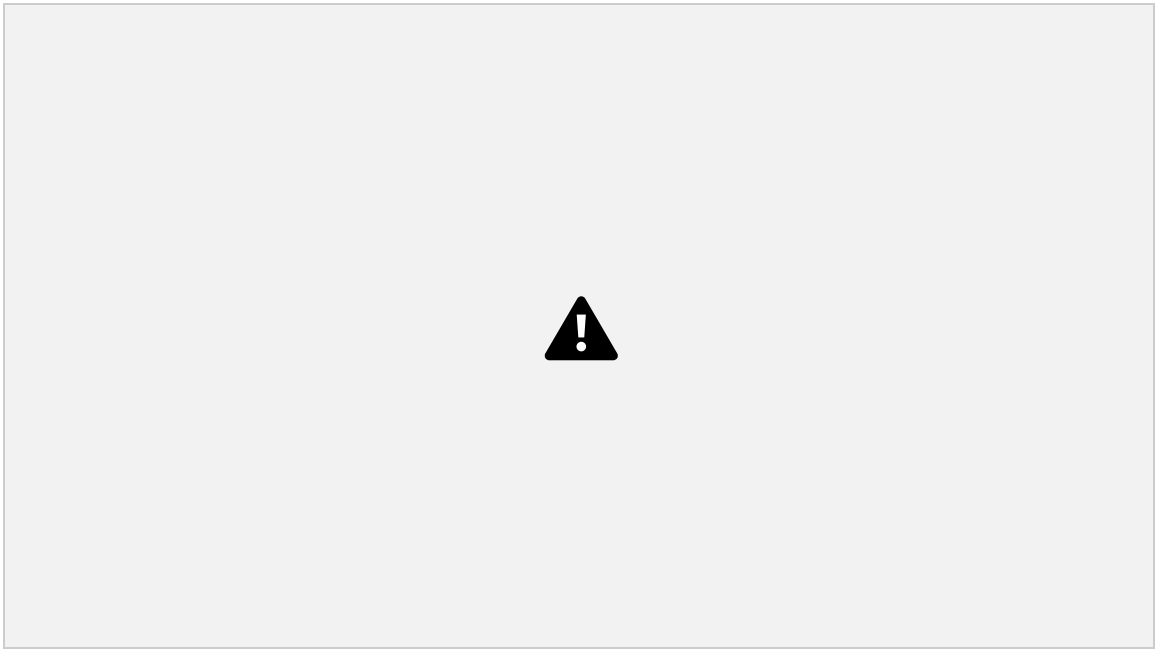
Types of Inheritance

1. **Single inheritance**
2. **multi-level inheritance**
3. **Hierarchical inheritance**
4. **Hybrid inheritance :It is the combination of the above 3 inheritance.**

Note: Multiple inheritance is not supported in java.

Multiple inheritance : One sub class having more than one parent class.





- **Single Inheritance: One class is inherited from one super class**

```
class X
{
}
class Y extends X
{
}
```

- **Multi-level Inheritance : One class is inherited from another sub class**

```
class Z extends Y
{
}
```

- **Hierarchycal Inheritance : More than one sub class inherited from one super class.**

```
class Y extends X
{
}
class Z extends X
{
}
```

```
class A{
void show_A(){System.out.println("Hello");}
}
class B extends A{
void show_B(){System.out.println("Welcome");}
}
```

```
 public static void main(String args[]){
   B obj=new B();
   obj.show_A();
 obj.show_B();

}
}
```

output

Hello

Welcome

One class can not extend more than one super class

Exa

```
class A{
void show(){System.out.println("Hello");}
}
class B{
void show(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 public static void main(String args[]){
   C obj=new C();
   obj.show();//Now which msg() method would be invoked?
}
}
```

o/p

It will give compilation error

Method Overriding in Java

- **Using method overriding, a subclass can change the behavior of inherited methods to meet its specific needs. It allows a programmer to implement methods in a derived class that has the same signature as in the base class. Same signature means the same name, the same number of parameters, and their types appearing in the same order.**
- **You are not allowed to override a method that has the final modifier.**
- **The method of a base class is known as overridden method, and the derived class method is known as overriding method.**
- **The return type of overriding and overridden method must be same.**
- **You are not allowed to make an overridden method less accessible. For example- if you are overriding a public method then you cannot declare it private, default, and protected.**

Example

```
class A

{

 public void show()

{

 System.out.println("show method of A");

}

void show(String msg) //overloaded show() method

{

  System.out.println(msg);

}


}

class V extends A

{
```

```
void show() //overriden show() method
{
 super.show();
 System.out.println("show of V");
}

}
class methoverride
{
public static void main(String[] s)
{
V o=new V();
o.show();
o.show("cime");
}
}
```

o/p compilation error

show() method in class V is default, we need to declare it public

Exa

```
public class Animal
{
        public void sound()
        {
                System.out.println("Animal makes sound");
```

```
        }
}
class Dog extends Animal
{
        public void sound()
        {
                System.out.println("Dog barks");
        }
        public static void main(String args[])
        {
                Dog d = new Dog();
                d.sound();
        }
}
```

Overriding of static method

- A static method of a super class cannot be overridden, but a derived class can define a method with the same signature. This will hide base class static method, and you can't even call it with super keyword.
- static methods inherited from a base class cannot be made non-static.
- Call base class static method with super keyword will give compile time error.

```
class Parent
{
        public static void show()
        {
                System.out.print("Parent");
        }
```

```
}
class Child extends Parent
{
        public static void show()
        {
                super.show(); //compilation error
                System.out.print("Child");
        }
        public static void main(String args[])
        {
                Child c = new Child();
                c.show();
        }
}
```

- Making base class static method ;non-static in a derived class will also give compile time error.

```
class Parent
{
        public static void show()
        {
                System.out.print("Parent");
        }
}
class Child extends Parent
{
```

```
    public void show()
    {
        System.out.print("Child");
    }
    public static void main(String args[])
    {
        Child c = new Child();
        c.show();
    }
}
```

Difference between Method Overriding and Overloading

Method Overriding

- In method overriding, the signature of overriding method must be same as that of overridden method. In other words, the number of arguments, their types, and the order must be same in both overriding and overridden methods.
- The return type of both overriding and overridden method must be same.
- It is an example of run-time polymorphism or dynamic polymorphism.
- You cannot make overriding method less public than the overridden method.

Method Overloading

- In method overloading, we have more than one functions with the same name but with different sets of argument types or the same set of argument types in different order.
- The return type of overloaded functions may be same or different.
- It is an example of compile-time polymorphism or static polymorphism.
- Overloaded functions can have different access modifiers.