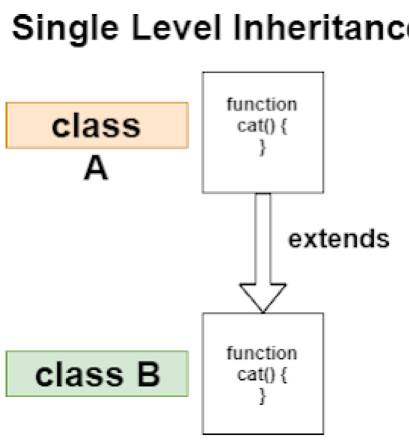
Inheritance in Java

It is the mechanism in Java by which one class is allowed to inherit the features(fields and methods) of another class. In Java, Inheritance means creating new classes based on existing ones. A class that inherits from another class can reuse the methods and fields of that class. In addition.

Why Do We Need Java Inheritance?

- **Code Reusability:** The code written in the Superclass is common to all subclasses. Child classes can directly use the parent class code.
- **Method Overriding: Method Overriding** is achievable only through Inheritance. It is one of the ways by which Java achieves Run Time Polymorphism.
- **Abstraction:** The concept of abstract where we do not have to provide all details is achieved through inheritance. **Abstraction** only shows the functionality to the user.



```
class A{
}


class B extends A{
}
```

Parent Class: The class that shares the fields and methods with the child class is known as parent class, super class or Base class. In the above code, Class A is the parent class.

Child Class: The class that extends the features of another class is known as child class, sub class or derived class. In the above code, class B is the child class.

Terminologies used in Inheritance:

- Super class and base class are synonyms of Parent class.
- Sub class and derived class are synonyms of Child class.
- Properties and fields are synonyms of Data members.
- Functionality is a synonym of method.

Extends: In Java, the *extends* keyword is used to indicate that the **class** which is being defined is derived from the base class using inheritance.

So basically, extends keyword is used to extend the functionality of the parent class to the subclass.

```
package Inheritance;

public class Teacher {

  // fields of parent class
  String designation = "Teacher";
  String collegeName = "ITM";

  // method of parent class
  void profession() {
    System.out.println("Teaching");
  }

}
```

```
package Inheritance;

public class ComputerScTeacher extends Teacher {

  String mainSubject = "CompSc";

  public static void main(String args[]) {
    ComputerScTeacher obj = new ComputerScTeacher();
    // accessing the fields of parent class
    System.out.println(obj.collegeName);
    System.out.println(obj.designation);

    System.out.println(obj.mainSubject);

    // accessing the method of parent class
    obj.profession();


  }
}
```

Based on the above example we can say that `CompscTeacher` **IS-A** `Teacher`. This means that a child class has **IS-A** relationship with the parent class. This is why inheritance is known as **IS-A relationship** between child and parent class

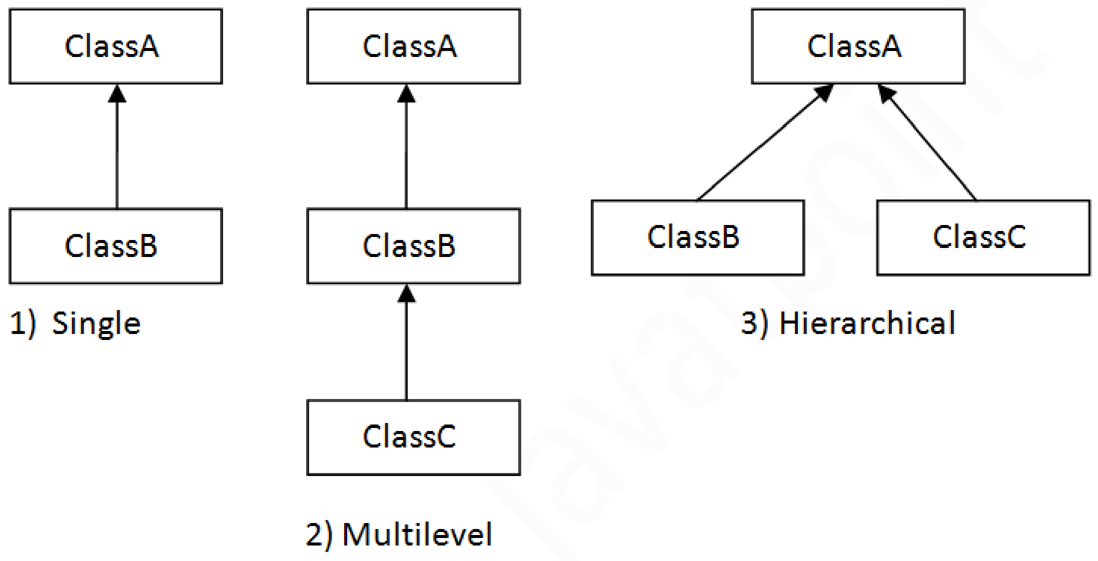
IS-A is a way of saying: **This object is a type of that object**.

```
public class Animal {
}
public class Mammal extends Animal {
}
public class Reptile extends Animal {
}
public class Dog extends Mammal {
}
```

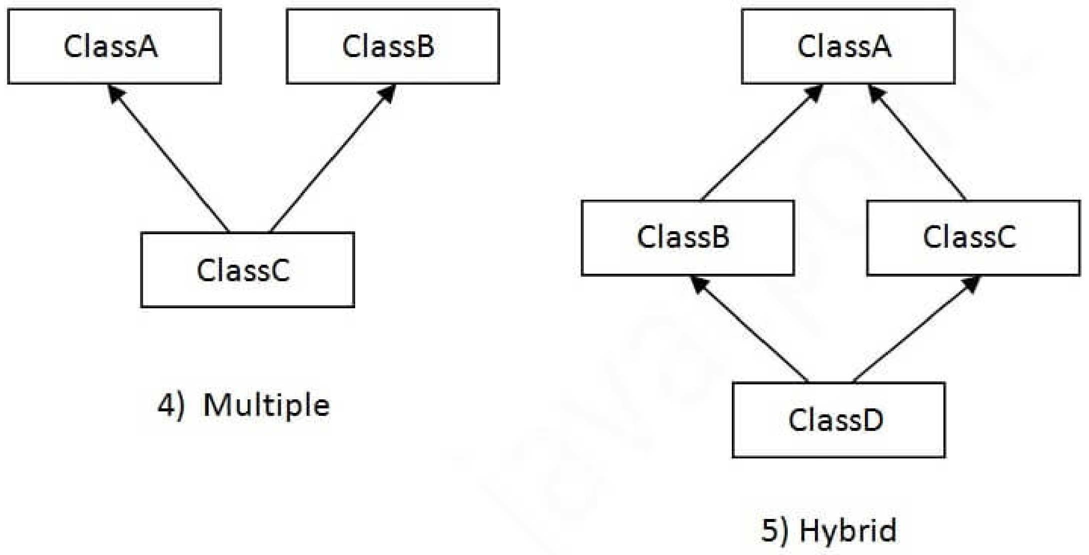
Now, if we consider the IS-A relationship

- Mammal IS-A Animal
- Reptile IS-A Animal
- Dog IS-A Mammal
- Hence: Dog IS-A Animal as well

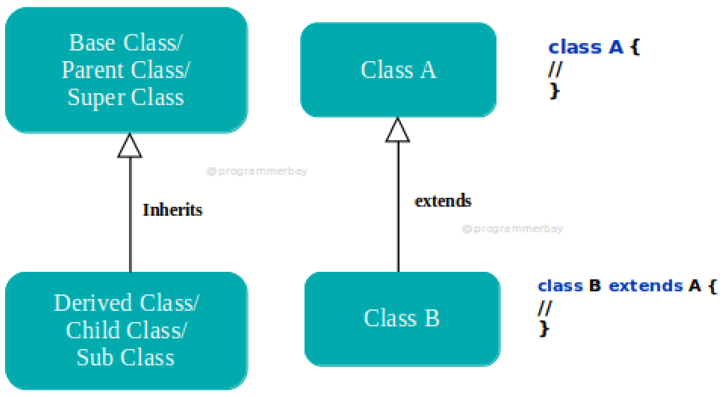
Types of inheritance



Implement By Interface

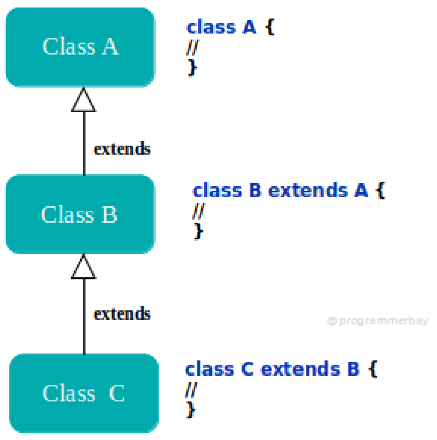


Single inheritance can be defined as a type of inheritance, where a single parent class is inherited by only a single child class.



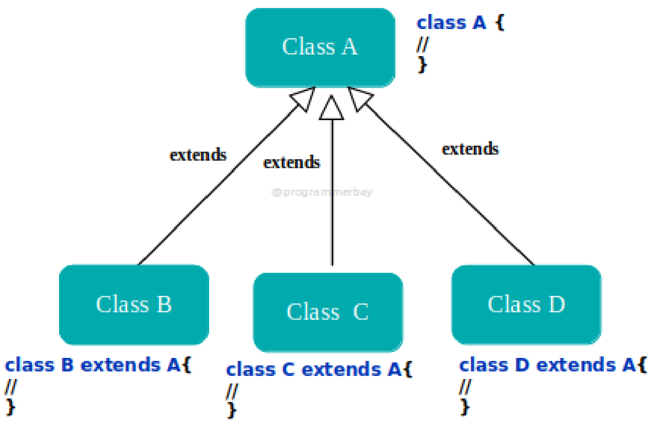
```
public class A {
    public void display() {
        System.out.println("I am a method from class A");
    }
}

// B is inheriting display method of A
class B extends A {
    public void print() {
        System.out.println("I am a method from class B");
    }

    public static void main(String[] args) {
        B objB = new B();
        objB.display(); // Reusing the method of A named display
        objB.print();
    }
}
```

Multilevel inheritance is a type of inheritance where a subclass acts as a superclass of another class. In other words, when a class having a parent class, is extended by another class and forms a sequential chain, then it's termed Multilevel inheritance.



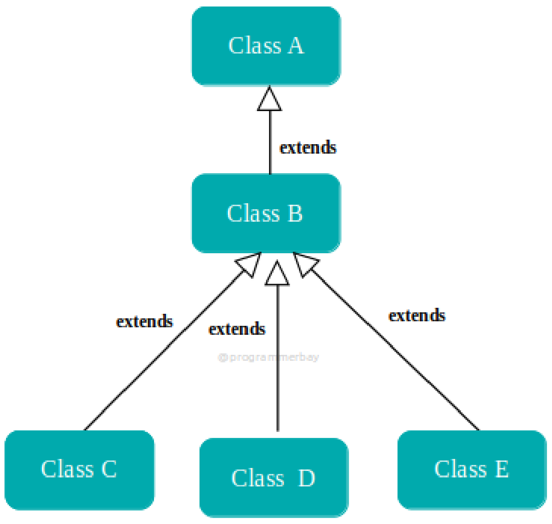
```
class A {
public void display() {
        System.out.println("I am a method from class A");
    }

}

class B extends A {
public void display() {
        System.out.println("I am a method from class B");
    }

}

class C extends B {
public void display() {
        System.out.println("I am a method from class C");
    }
}
```

Hierarchical inheritance is a type of inheritance in which two or more classes inherit a single parent class. In this, multiple classes acquire properties of the same superclass. The classes that inherit all the attributes or behaviour are known as child classes or subclass or derived classes. The class that is inherited by others is known as a superclass or parent class or base class. For instance, class B, class C, and class D inherit the same class name.



```
public class A {
    public void display() {
        System.out.println("I am a method from class A");
    }
}

class B extends A {
    public void print() {
        System.out.println("I am a method from class B");
    }
}

class C extends A {
    public void show() {
        System.out.println("I am a method from class C");
    }
}

class D extends A {
    public void outPut() {
        System.out.println("I am a method from class D");
    }

    public static void main(String[] args) {
        B objB = new B();
        C objC = new C();
        D objD = new D();
        objB.display();
        objC.display();
        objD.display();
    }
}
```

Hybrid inheritance is a type of inheritance in which two or more variations of inheritance are used. For instance, suppose, there are various classes namely A, B, C, D and E. if class A gets extended by class B, then this would be an example of Single inheritance. Further, if class C, D and E extend class B,then it would be an example of hierarchical inheritance. In this manner, multiple types of inheritance can be used within the same hierarchy structure, that is termed Hybrid inheritance.



```
public class A {
    public void display() {
        System.out.println("I am a method from class A");
    }
}

class B extends A {
    public void print() {
        System.out.println("I am a method from class B");
    }
}

class C extends B {
    public void show() {
        System.out.println("I am a method from class C");
    }
}

class D extends C {
    public void show() {
        System.out.println("I am a method from class D");
    }
    public static void main(String[] args) {
        D objD = new D();
        objD.display(); // A is indirect parent of D, therefore, the display method gets inherited all the way to leaf
    }
}
```