Java – *Inheritance*:

- The process, where one class acquires the properties (methods and fields) of another. / Allows a class to use the properties and methods of another class.
- In other words, the derived class **inherits** the states and behaviors from the base class. The derived class is also called subclass and the base class is also known as super-class.
- Used to manage the information in a hierarchical order.
- The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).
- **extends** is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

Syntax:

```
class Super {
  .....
  .....
}
class Sub extends Super {
  .....
  .....
}
```

Oracle – java Documentation

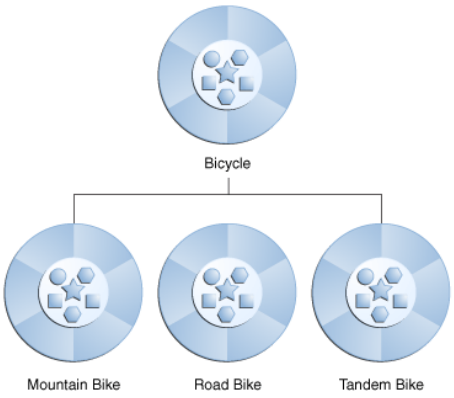
Definitions: A class that is derived from another class is called a *subclass* (also a *derived class*, *extended class*, or *child class*). The class from which the subclass is derived is called a *superclass* (also a *base class* or a *parent class*).

Excepting Object, which has no superclass, every class has one and only one direct superclass (single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of Object.

Classes can be derived from classes that are derived from classes that are derived from classes, and so on, and ultimately derived from the topmost class, Object. Such a class is said to be *descended* from all the classes in the inheritance chain stretching back to **Object**.

The idea of inheritance is simple but powerful: When you want to create a new class and there is already a class that includes some of the code that you want, you can derive your new class from the existing class. In doing this, you can ***reuse the fields and methods of the existing class without having to write*** (and debug!) them yourself.

A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.



The syntax for creating a subclass is simple. At the beginning of your class declaration, use the `extends` keyword, followed by the name of the class to inherit from:

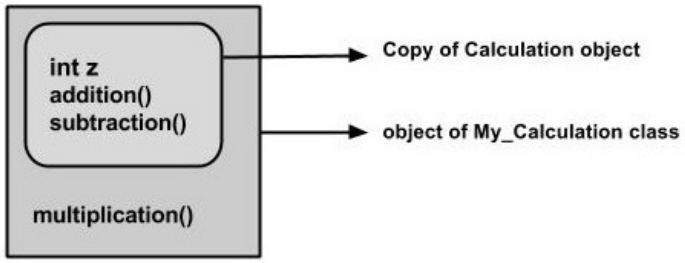
```
class MountainBike extends Bicycle {

    // new fields and methods defining
    // a mountain bike would go here

}
```

This gives `MountainBike` all the same fields and methods as `Bicycle`, yet allows its code to focus exclusively on the features that make it unique. This makes code for your subclasses easy to read. However, you must take care to properly document the state and behavior that each superclass defines, since that code will not appear in the source file of each subclass.

Example-1:

```
class Calculation {
   int z;

   public void addition(int x, int y) {
      z = x + y;
      System.out.println("The sum of the given numbers: "+z);
   }

   public void Subtraction(int x, int y) {
      z = x - y;
      System.out.println("The difference between the given numbers: "+z);
   }
}

public class My_Calculation extends Calculation {
   public void multiplication(int x, int y) {
      z = x * y;
      System.out.println("The product of the given numbers: "+z);
   }

   public static void main(String args[]) {
      int a = 20, b = 10;
      My_Calculation demo = new My_Calculation();
      demo.addition(a, b);
      demo.Subtraction(a, b);
      demo.multiplication(a, b);
   }
}
```

Output: ???



Note: A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

The super keyword

The **super** keyword is similar to **this** keyword. Following are the scenarios where the super keyword is used.

- It is used to **differentiate the members** of superclass from the members of subclass, if they have same names.
- It is used to **invoke the superclass** constructor from subclass.

Differentiating the Members

If a class is inheriting the properties of another class. And if the members of the superclass have the names same as the sub class, to differentiate these variables we use super keyword as shown below.

```
super.variable
super.method();
```

Example-2:

```
class Super_class {
   int num = 20;

  // display method of superclass
   public void display() {
      System.out.println("This is the display method of superclass");
   }
}

public class Sub_class extends Super_class {
   int num = 10;

  // display method of sub class
   public void display() {
      System.out.println("This is the display method of subclass");
   }

   public void my_method() {
      // Instantiating subclass
      Sub_class sub = new Sub_class();

         sub.display();   // Invoking the display() method of sub class

         super.display();   // Invoking the display() method of superclass

           System.out.println("value of the variable named num in sub
class:"+ sub.num);   // printing the value of variable num of subclass


      System.out.println("value of the variable named num in super
class:"+ super.num);   // printing the value of variable num of superclass
   }

   public static void main(String args[]) {
      Sub_class obj = new Sub_class();
      obj.my_method();
   }
}
```

Output: ???

Invoking Superclass Constructor

If a class is inheriting the properties of another class, the subclass automatically acquires the default constructor of the superclass. But if you want to call a parameterized constructor of the superclass, you need to use the super keyword as shown below.

super(values);

Example-3:

```
class Superclass {
   int age;

   Superclass(int age) {
      this.age = age;
   }

   public void getAge() {
      System.out.println("The value of the variable named
age in super class is: " +age);
   }
}

public class Subclass extends Superclass {
   Subclass(int age){
      super(age);
   }

   public static void main(String argd[]) {
      Subclass s = new Subclass(24);
      s.getAge();
   }
}
```

Output: ???

Example-4:

```
class Vehicle{
    String vehicleType;
}
public class Car extends Vehicle{

    String modelType;
    public void showDetail(){
        vehicleType = "Car"; //accessing Vehicle class member
        modelType = "sports";
        System.out.println(modelType+" "+vehicleType);
    }
    public static void main(String[] args){
        Car car =new Car();
        car.showDetail();
    }
}
```

Output: ???

Example-4: Child class calling Parent class constructor using super keyword

```
class Parent{
    String name;

    public Parent(String n){
        name = n;
    }

}
public class Child extends Parent {
    String name;

    public Child(String n1, String n2){

        super(n1); //passing argument to parent class constructor
        this.name = n2;
    }
    public void details() {
        System.out.println(super.name+" and "+name);
    }
     public static void main(String[] args){
        Child cobj = new Child("Parent","Child");
        cobj.details();
    }
}
```