Inheritance

- Inheritance is a mechanism in which one object acquires all the properties and behaviors of parent object. In other words, you can create new classes from existing classes, then you can reuse methods and variables of parent class, and you can add new methods and variables also.

- Inheritance is an important pillar of OOP (Object-Oriented Programming).

Important terminology:

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

Benefits of Inheritance

- Inheritance helps in code reuse. The child class may use the code defined in the parent class without re-writing it.
- Inheritance can save time and effort as the main code need not be written again.

How to use inheritance in Java

The keyword used for inheritance is extends. **extends** keyword The extends keyword indicates that you are making a new class that derives from an existing class.

Syntax :

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

Example 1:

```
class Employee{
```

هنا في هذا المثال لدينا كلاس خاص بالموظفين يحتوي على الاسم والراتب و رقم الموظف

وكلاس اخر يحتوي يرث من الكلاس الأول ويضيف علية خاصية جديده واحده وبالتالي اصبح الكلاس يحتوي على أربعة خصائص

```
int E_no;
String full_name;
Double  salary; }
class  Edit_emmployee extends Employee{
int bonus=1000;
public static void main(String args[]){
 Edit_emmployee p=new Edit_emmployee () ;
p. full_name ="Ali";
p.E_no= 123;
System.out.print("Programmer Information :"+p. full_name + " "+ p.salary);
System.out.println(" "+ p.E_no + " "+ p.bonus);}
}
```

هنا نلاحظ تعريف كائن من الكلاس الثاني واحتواءه على خصائص من الكلاس الأول

Example2:

```
class compute{
public void add_two_number(int x, int y){
System.out.println("The sum of the given numbers:"+( x+y));
}
public void Sub_two_number(int x, int y){
System.out.println("The difference between the given numbers:"+ ( x-y));}
}
public class Edit_compute extends compute{
public void mult_two_number (int x, int y){
System.out.println("The product of the given numbers:"+ ( x*y));
}
public static void main(String args[]){
int a = 24, b = 6;
Edit_compute d=new Edit_compute ();
d. add_two_number (a, b);
```

هنا الكلاس الثاني ورث الدوال من الكلاس الأول وأضاف الدالة الخاصة به

d. **Sub_two_number** (a, b);
d**.mult_two_number** (a, b);}
}

Example3: In the below example of inheritance, class **Bicycle** is a base class, class **MountainBike** is a derived class that extends Bicycle class and class Test is a driver class to run program.

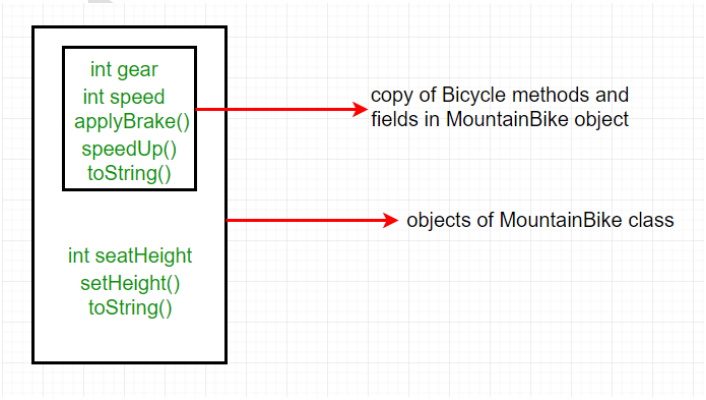
```
class Bicycle {
    int gear;
    int speed;
     public Bicycle(int gear, int speed)
    {
        this.gear = gear;
        this.speed = speed;
    }
     public void applyBrake(int decrement)
    {
        speed -= decrement;
    }

    public void speedUp(int increment)
    {
        speed += increment;
    }
    public String toString()
    {
        return ("No of gears are " + gear + "\n"
                + "speed of bicycle is " + speed);
    }
}
 class MountainBike extends Bicycle {
    int seatHeight;
    MountainBike(int gear, int speed, int startHeight)
```

```
    {
        super(gear, speed);
        seatHeight = startHeight;
    }
     public void setHeight(int newValue)
    {
        seatHeight = newValue;
    }
    @Override
    public String toString()
    {
        return (super.toString() + "\nseat height is "
                + seatHeight);
    }
}
 public class Test {
    public static void main(String args[])
    {
         MountainBike mb = new MountainBike(3, 100, 25);
        System.out.println(mb.toString());
    }
}
```

Output

No of gears are 3
speed of bicycle is 100
seat height is 25

In the above program, when an object of **MountainBike** class is created, **a copy of all methods and fields of the superclass acquire memory in this object.** That is why by using the object of the subclass we can also access the members of a superclass.



Notes:

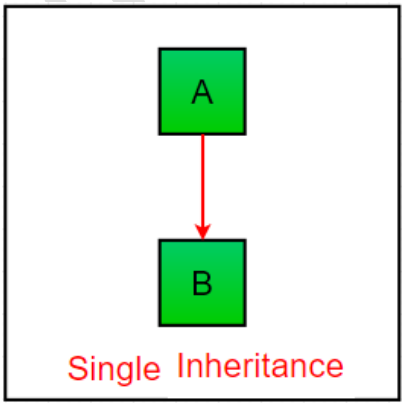
- A subclass inherits all the members (variables, methods) from its superclass.
- Constructors are not members, **so they are not inherited by subclasses**, but it can be invoked from the subclass.
- Members declared **private** are not inherited at all.

In practice, **inheritance** and **polymorphism** are used together in java to achieve fast performance and readability of code.

Types of Inheritance in Java

Below are the different types of inheritance which are supported by Java.

1. **Single Inheritance:** In single inheritance, subclasses inherit the features of one superclass. In the image below, class A serves as a base class for the derived class

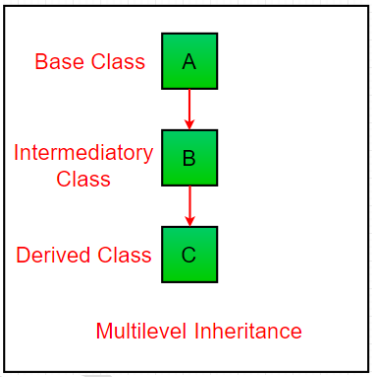



```
class one {
   public void print ()
   {
      System.out.println("Hello");
   }
}
class two extends one {
   public void print_two() { System.out.println("the two class"); }
}
// Driver class
public class Main {
   public static void main(String[] args)
   {
      two g = new two();
      g.print();
      g.print_two();
      g.print ();
   }
}
```

Output

Hello
the two class
Hello

2- **Multilevel Inheritance**: In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In the below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access the grandparent's members.



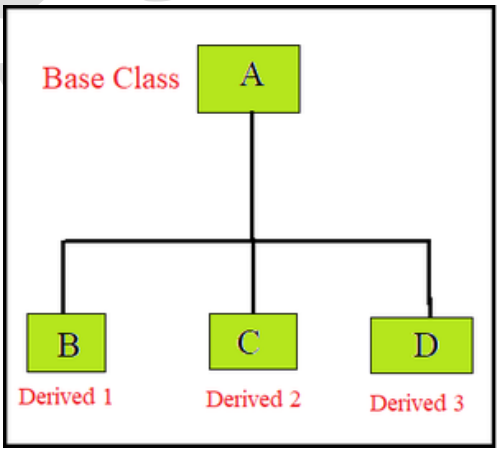

```
class one {
   public void print_one()
   {
      System.out.println("one");
   }
}
class two extends one {
   public void print_two() { System.out.println("two"); }
}

class three extends two {
   public void print_three()
   {
      System.out.println("three");
   }
}
public class Main {
   public static void main(String[] args)
   {
      three g = new three();
      g.print_one();
      g.print_two();
      g.print_three();
   }
}
```

Output

one
two
three

3- **Hierarchical Inheritance**: In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass. In the below image, class A serves as a base class for the derived class B, C and D.

```
class A {
   public void print_A() { System.out.println("Class A"); }
}
class B extends A {
   public void print_B() { System.out.println("Class B"); }
}
class C extends A {
   public void print_C() { System.out.println("Class C"); }
}
class D extends A {
   public void print_D() { System.out.println("Class D"); }
}

public class Test {
   public static void main(String[] args)
   {
      B obj_B = new B();
      obj_B.print_A();
      obj_B.print_B();

      C obj_C = new C();
      obj_C.print_A();
      obj_C.print_C();

      D obj_D = new D();
      obj_D.print_A();
      obj_D.print_D();
   }
}
```



Output

Class A
Class B
Class A
Class C
Class A
Class D

4. **Multiple Inheritance** (Through Interfaces): In Multiple inheritances, one class can have more than one superclass and inherit features from all parent classes. **Please note that Java does not support multiple inheritances with classes**. In java, we can achieve multiple inheritances only through Interfaces.

Why Java does not support multiple inheritances with classes?

Ans:

5**. Hybrid Inheritance**(Through Interfaces): It is a mix of two or more of the above types of inheritance. Since java doesn't support multiple inheritances with classes, **hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through Interfaces.**

Important facts about inheritance in Java

Superclass can only be one: A superclass can have any number of **subclasses**. **But a subclass can have only one superclass**. This is because Java does not support multiple inheritances with classes. Although with interfaces, multiple inheritances are supported by java.

Inheriting Constructors: 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.**

Private member inheritance: A subclass does not inherit the private members of its parent class. However, if the superclass has public or protected methods(like getters and setters) for accessing its private fields, these can also be used by the subclass.

Super Keyword in Java: The super keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer **parent class instance variable.**
2. super can be **used to invoke parent class method.**
3. super() can be used to invoke **parent class constructor**.

Example for: super is used to refer parent class instance variable.

We can use super keyword to access the data member or field of parent class. **It is used if parent class and child class have same fields.**

```
class Animal{
String color="white";
}
class cat extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of cat class
System.out.println(super.color);//prints color of Animal class
}
}
class TestSuper1{
public static void main(String args[]){
cat c=new cat();
c.printColor();
}}
```

In the above example, Animal and cat both classes have a **common property color**. If we print color property, it will print the color of current class by default. To access the parent property, we need to use **super** keyword.

Example for: super can be used to invoke **parent class method**

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if **method is overridden**.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Cat extends Animal{
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
super.eat();
bark();
}
}
class TestSuper2{
public static void main(String args[]){
Cat c=new Cat();
c.work();
}}
```

Output:

eating...

barking...

In the above example **Animal and Cat both classes have eat() method** if we call **eat()** method from Cat class, it will call the eat() method of Cat class **by default because priority is given to local.** To call the parent class method, we need to use **super** keyword.

Example for: super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor. Let's see a simple example:

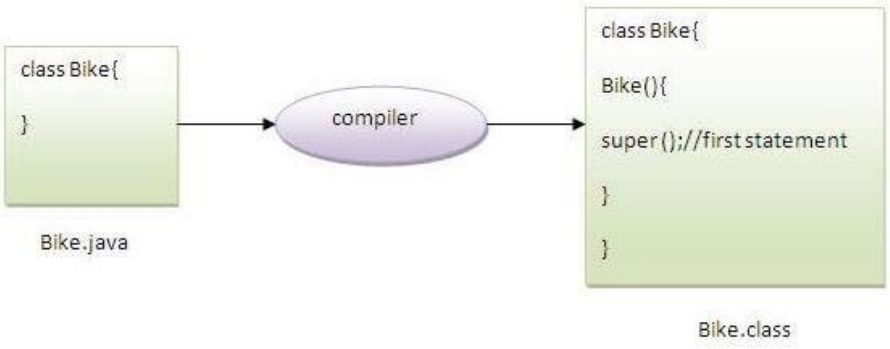
```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Cat extends Animal{
Cat(){
super();
System.out.println("Cat is created");
}
}
class TestSuper3{
public static void main(String args[]){
Cat d=new Cat();
}}
```

Output:

animal is created

Cat is created

Note: super() is added in each class constructor automatically by compiler if there is no super() .



As we know well that default constructor is provided by compiler automatically if there is no constructor. But, **it also adds super() as the first statement**.