Inheritance in java

- Inheritance is inheriting the properties of parent class into child class.
- Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of a parent object.
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class.
- you can also add new methods and fields in your current class.
- Inheritance represents the IS_A relationship which is also known as parent-child relationship.

  Eg:

  Dog IS_A Animal

  Car IS_A Vehicle

  Employee IS_A Person

  Surgeon IS_A Doctor etc.

```
class Animal
{
public  void eat()
{
}
}
```

```
Class Dog extends Animal
{
Public static void main(String args[])
{
Dog d=new Dog;
d.eat();
}
}
```

Syntax of Java Inheritance

```
class <Subclass-name>  extends  <Superclass-name>
{
  //methods and fields
}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

Eg_1:

```
import java.io.*;
class Person {

int id;
String name;
void set_Person()
{
   try{
                                        BufferedReader      br=new      BufferedReader(new
InputStreamReader(System.in));
      System.out.println("Enter the Id:");
```

```
        id=Integer.parseInt(br.readLine());
        System.out.println("Enter the Name");
        name=br.readLine();

    }catch(Exception ex){ex.printStackTrace();}
}

void disp_Person()
{
    System.out.print(id+"\t"+name+"\t");
}
}
```

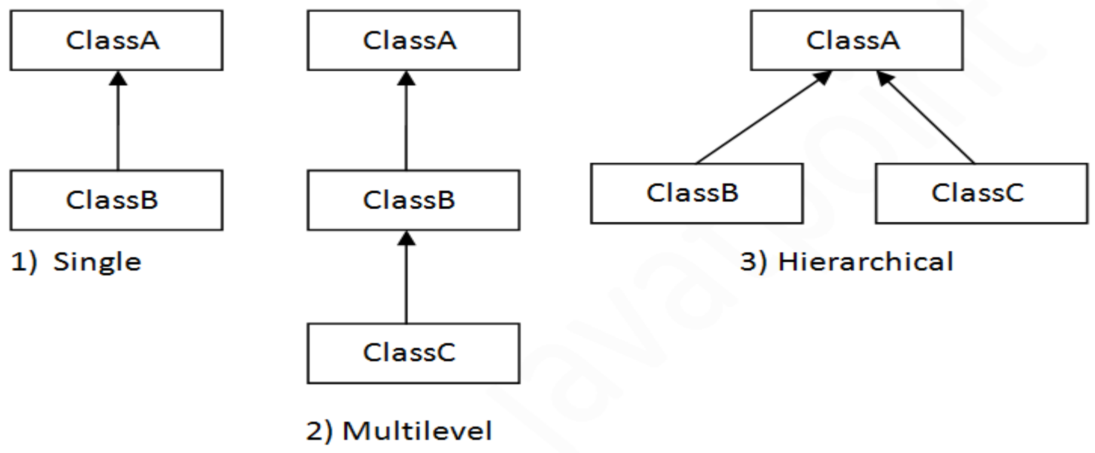
```
class Employee extends Person{
    int sal;
String desgn;
void set_Emp()
{
    try{
        set_Person();
                                        BufferedReader     br=new     BufferedReader(new
InputStreamReader(System.in));
        System.out.println("Enter the Designation:");
        desgn=br.readLine();
        System.out.println("Enter the Salary:");
        sal=Integer.parseInt(br.readLine());


    }catch(Exception ex){ex.printStackTrace();}
}

void disp_Emp()
{
    disp_Person();
    System.out.println(desgn+"\t"+sal);
}

public static void main(String args[])
{
    Employee e1=new Employee();
    e1.set_Emp();
    e1.disp_Emp();
}
}
```

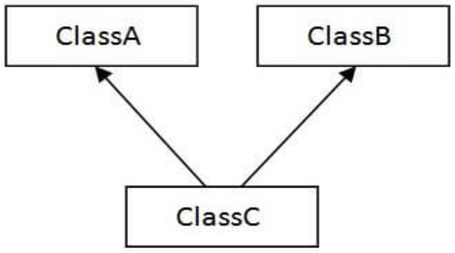
Eg_2:

```
class Person1 {
   int id;
String name;
void set_Person(int id,String name)
{
   try{

      this.id=id;
      this.name=name;

   }catch(Exception ex){ex.printStackTrace();}
}

void disp_Person()
{
   System.out.print(id+"\t"+name+"\t");
}
}

class Employee1 extends Person1 {
    int sal;
String desgn;
void set_Emp(int id,String name,String desgn,int sal)
{
   try{
      set_Person(id,name);
     this.desgn=desgn;
     this.sal=sal;


   }catch(Exception ex){ex.printStackTrace();}
}

void disp_Emp()
{
   disp_Person();
   System.out.print(desgn+"\t"+sal);
}

public static void main(String args[])
{
   Employee1 e1=new Employee1();
   e1.set_Emp(1001,"Manjeet","AP",20000);
   e1.disp_Emp();
}
}
```

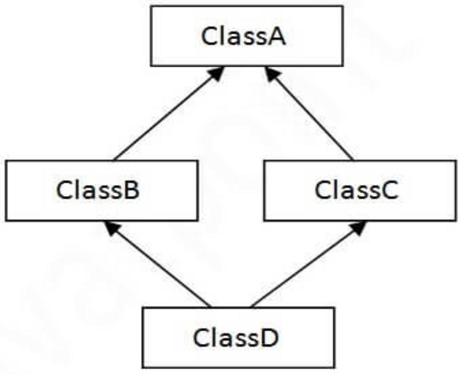
Types of inheritance in java

- Java Support three types of inheritance in java: single level, multilevel and hierarchical inheritance in case of classes to avoid ambiguity.
- In java programming, multiple and hybrid inheritance is supported through interface only.





4) Multiple



5) Hybrid

Single Inheritance Example

When a class inherits another class, it is known as a *single inheritance*.

```
class A
{
   int a;
   void set_A(int x)
   {
      a=x;
   }

}

 class B extends A{

   int b,product;
   void set_B(int x)
   {
      b=x;
   }
   void cal_Product()
   {
      product=a*b;
      System.out.println("Product ="+product);
   }
```

```
    public static void main(String[] args) {
        B b=new B();
        b.set_A(5);
        b.set_B(5);
        b.cal_Product();
    }
}
```

Multilevel Inheritance Example

When there is a chain of inheritance, it is known as *multilevel inheritance*.

```
class A
{
   int a;
   void set_A(int x)
   {
      a=x;
   }

}

 class B extends A{
   int b;
   void set_B(int x)
   {
      b=x;
   }

}

class C extends B{
   int c,product;

   void cal_Product()
   {
      product=a*b;
      System.out.println("Product ="+product);
   }

   public static void main(String[] args) {
       C c=new C();
       c.set_A(5);
       c.set_B(5);
       c.cal_Product();
    }
}
```

Hierarchical Inheritance Example

When two or more classes inherit a single class, it is known as hierarchical inheritance.

Eg:

```
class A
{
   int a;
   void set_A(int x)
   {
      a=x;
   }

}

class B extends A{
   int b;
   void set_B(int x)
   {
      b=x;
   }

}

class C extends A{
   int c;
   void set_C(int x)
   {
      c=x;
   }

}
```

Method Overriding in Java

- If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.
- When a method in a subclass has the same name, same parameters or signature and same return type as a method in its super-class, then the method in the subclass is said to override the method in the super-class.
- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism.
- It must be an IS-A relationship (inheritance).

Eg:

```
//Java Program to illustrate the use of Java Method Overriding
//Creating a parent class.
class Vehicle{
 //defining a method
 void run()        // overridden method
 {
    System.out.println("Vehicle is running");
  }
}
//Creating a child class
class Bike extends Vehicle{
 //defining the same method as in the parent class
 void run()      //overriding method
 {
    System.out.println("Bike is running safely");
 }

 public static void main(String args[]){
 Bike obj = new Bike();//creating object
 obj.run();//calling method
 }
}
```

Eg:

```
class Animal {
   public void move() {
      System.out.println("Animals can move");
   }
}

class Dog extends Animal {
   public void move() {
      System.out.println("Dogs can walk and run");
   }
}

class Test {

   public static void main(String args[]) {
      Animal a = new Animal();   // Animal reference and Animal object
      Animal b = new Dog();   // Animal reference but Dog object


      a.move();   // runs the method in Animal class

      b.move();   // runs the method in Dog class
   }
}
```

Output:

```
Animals can move
Dogs can walk and run
```

// When invoking a superclass version of an overridden method the **super** keyword is used.

The “super” keyword in java

- **super** keyword in java is a reference variable can be used to refer immediate parent class instance variable.
- Super keyword can be used to invoke immediate parent class method.
- super() can be used to invoke immediate parent class constructor.

1) super keyword is used to refer immediate 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 Dog extends Animal{

String color="black";

void printColor(){

System.out.println(color);//prints color of Dog class


System.out.println(super.color);//prints color of Animal class

}

}


class Test{

public static void main(String args[]){

Dog d=new Dog();

d.printColor();

}}
```

2. super keyword can be used to invoke immediate parent class method.

Eg_1:

```
import java.io.*;
class Person {

int id;
String name;
void set_data()
{
   try{
                                        BufferedReader     br=new     BufferedReader(new
InputStreamReader(System.in));
        System.out.println("Enter the Id:");
        id=Integer.parseInt(br.readLine());
        System.out.println("Enter the Name");
        name=br.readLine();
   }catch(Exception ex){ex.printStackTrace();}
}

void display()
{
   System.out.print(id+"\t"+name+"\t");
}
}


class Employee extends Person{
   int sal;
String desgn;
void set_data()
{
   try{
      super.set_data();
                                          BufferedReader     br=new     BufferedReader(new
InputStreamReader(System.in));
      System.out.println("Enter the Designation:");
      desgn=br.readLine();
      System.out.println("Enter the Salary:");
      sal=Integer.parseInt(br.readLine());


   }catch(Exception ex){ex.printStackTrace();}
}

void display ()
{
   super.display();
   System.out.println(desgn+"\t"+sal);
}

public static void main(String args[])
{
   Employee e1=new Employee();
   e1.set_data();
   e1.display();
}
}
```

Eg_2:

```
class Person1 {
   int id;
String name;
void set_data(int id,String name)
{
   try{

      this.id=id;
      this.name=name;

   }catch(Exception ex){ex.printStackTrace();}
}

void display()
{
   System.out.print(id+"\t"+name+"\t");
}
}

class Employee1 extends Person1 {
     int sal;
String desgn;
void set_data(int id,String name,String desgn,int sal)
{
   try{
      super.set_data(id,name);
     this.desgn=desgn;
     this.sal=sal;


   }catch(Exception ex){ex.printStackTrace();}
}

void display()
{
   super.display();
   System.out.print(desgn+"\t"+sal);
}

public static void main(String args[])
{
   Employee1 e1=new Employee1();
   e1.set_data(1001,"Manjeet","AP",20000);
   e1.disp_data();
}
}
```

3. super() can be used to invoke immediate parent class constructor.

Eg_1:

```
import java.io.*;
class Person {

int id;
String name;
public Person()
{
   try{
                                BufferedReader    br=new    BufferedReader(new
InputStreamReader(System.in));
      System.out.println("Enter the Id:");
      id=Integer.parseInt(br.readLine());
      System.out.println("Enter the Name");
      name=br.readLine();

   }catch(Exception ex){ex.printStackTrace();}
}

void display()
{
   System.out.print(id+"\t"+name+"\t");
}
}


class Employee extends Person{
   int sal;
String desgn;
public Employee()
{
   try{
      super();
```

```
                                        BufferedReader    br=new    BufferedReader(new
InputStreamReader(System.in));
        System.out.println("Enter the Designation:");
        desgn=br.readLine();
        System.out.println("Enter the Salary:");
        sal=Integer.parseInt(br.readLine());


    }catch(Exception ex){ex.printStackTrace();}
}

void display ()
{
    super.display();
    System.out.println(desgn+"\t"+sal);
}

public static void main(String args[])
{
    Employee e1=new Employee();
    e1.display();
}
}
```

Eg_2:

```
class Person1 {
   int id;
String name;
public Person1(int id,String name)
{
   try{

      this.id=id;
      this.name=name;

   }catch(Exception ex){ex.printStackTrace();}
}

void display()
{
   System.out.print(id+"\t"+name+"\t");
}
}

class Employee1 extends Person1 {
     int sal;
String desgn;
public Employee1(int id,String name,String desgn,int sal)
{
   try{
      super(id,name);
     this.desgn=desgn;
     this.sal=sal;


   }catch(Exception ex){ex.printStackTrace();}
}

void display()
{
   super.display();
   System.out.print(desgn+"\t"+sal);
}

public static void main(String args[])
{
    Employee1 e1=new Employee1(1001,"Manjeet","AP",20000);
    e1.disp_data();
}
}
```

Difference between super keyword and super method in java

super	**super()**
The **super keyword** in **Java** is a reference variable that is used to refer **parent** class objects.	The **super**() in **Java** is used to refer **parent** class constructors.

Final Keyword in Java

The **final keyword** in java is used to restrict the user. The Java final keyword can be used in many contexts. **final keyword** can be:

1. variable
2. method
3. class

1) Java final variable

If you make any variable as final, you cannot change the value of final variable (It will be constant).

Example of final variable

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Bike{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike obj=new  Bike();
 obj.run();
 }
}//end of class
```

Output:`Compile Time Error`

2) Java final method

If you make any method as final, you cannot override it.

Example of final method

```
class Bike{
  final void run(){System.out.println("running");}
}

class Honda extends Bike{
   void run(){System.out.println("running safely with 100kmph");}

   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}
```

Output:`Compile Time Error`

3) Java final class

If you make any class as final, you cannot extend it.

Example of final class

```
final class Bike{}

class Honda extends Bike{
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda honda= new Honda();
  honda.run();
  }
}
```

```
Output:Compile Time Error
```

Q) Is final method inherited?

Ans) Yes, final method is inherited but you cannot override it. For Example:

```
class Bike{
  final void run(){System.out.println("running...");}
}
class Honda extends Bike{
   public static void main(String args[]){
    new Honda().run();
   }
}
```