

Inheritance (IS-A relationship) in Java

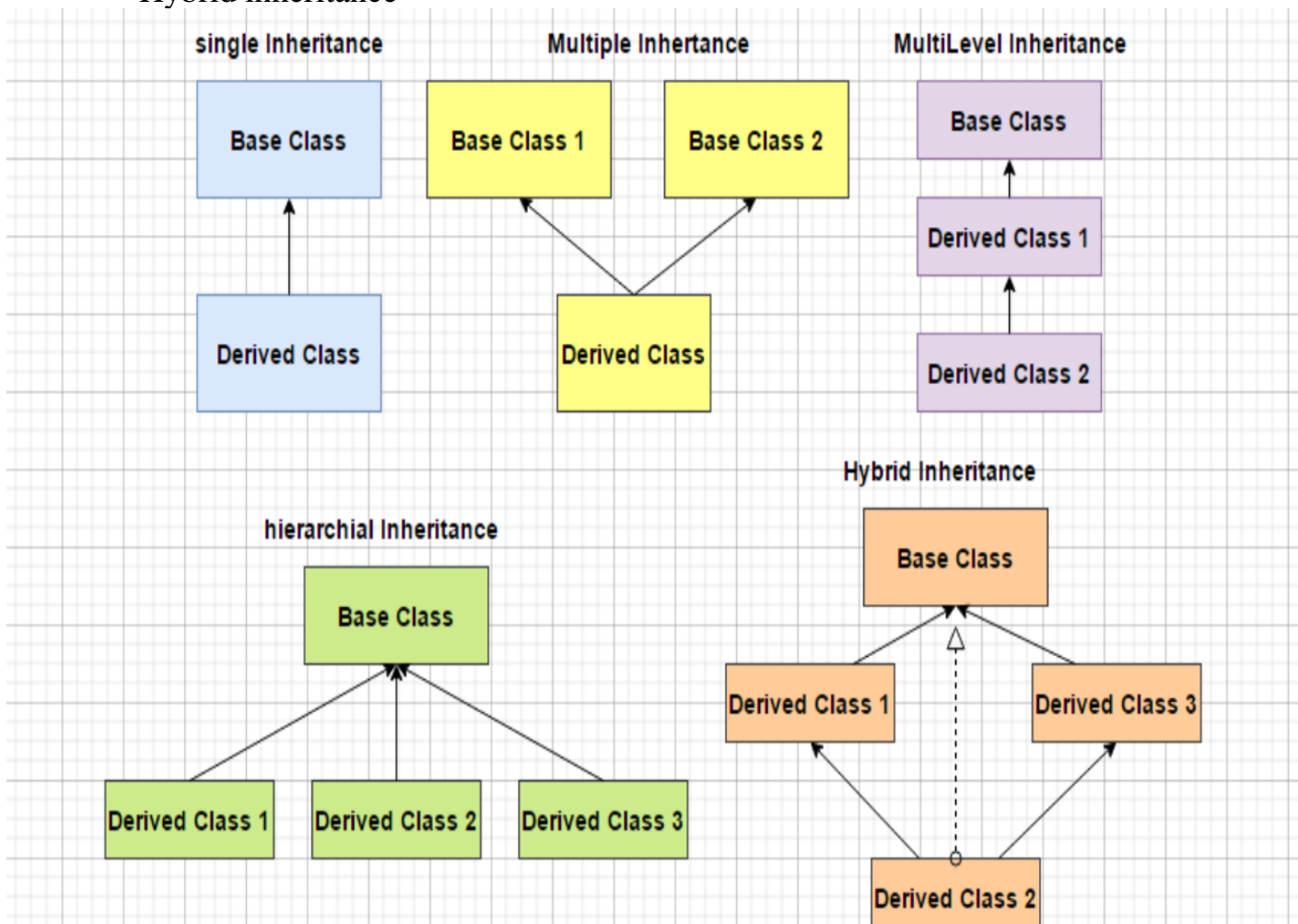
Inheritance defines is-a relationship between a Super class and its Sub class.

Advantages of Inheritance: It promotes the code **reusability** and run time **polymorphism**.


Disadvantages of Inheritance: Tightly coupled.

JAVA supports four types of inheritance:

- Single inheritance
- **Multiple inheritance** (Multiple inheritance is not supported by java class)
- Hierarchical inheritance
- Multilevel inheritance
- Hybrid inheritance



- **Single Level Inheritance**

Class A { STM(s); } Class B extends A { STM(s); }	
--	---

Example:

```

class A
{
    int i=10;
    A() {System.out.println("i="+i);}
}

class B extends A
{
    int j=20;
    B() {System.out.println("j="+j);}
}

class SingleLevelInheritanceDemo
{
    public static void main(String arg[])
    {
        new A();
        new B();
    }
}

```

Example2

```

class Parent
{
    public void method1()
    {
        System.out.println("Parent method invoked");
    }
}

class Child extends Parent {

    public void method2()
    {
        System.out.println("Child method invoked");
    }
}


```

```

    }
}
class SingleLevelInheritanceDemo
{
    public static void main(String[] args)
    {
        Child cobj = new Child();
        cobj.method1();
        cobj.method2();
    }
}

```

Multilevel inheritance

<pre> Class A { STM(s); } Class B extends A { STM(s); } Class C extends B { STM(s); } </pre>	 <pre> graph BT A[A] B[B] --> A C[C] --> B </pre>
--	---

Example

```

class A
{
    int i=10;
    A() {System.out.println("i="+i);}
}
class B extends A
{
    int j=20;
    B() {System.out.println("j="+j);}
}
class C extends B
{
    int k=10;    // k=30
    C() {System.out.println("k="+k);}
}
class MultilevelInheritanceDemo
{

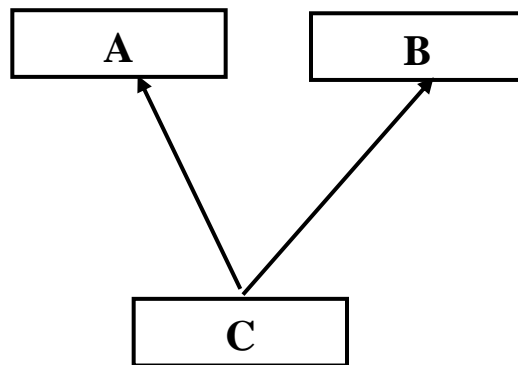
```

```

public static void main(String arg[])
{
    new A();
    new B();
    new C();
}

```

O/P: i=10
 i=10
 j=20
 i=10
 j=20
 k=10



Multiple Inheritance

```

class A
{
    void method() {System.out.println("method in A");}
}
class B
{
    void method() {System.out.println("method in B");}
}
class C extends B,A
{
    void mehod2() {System.out.println("method2 in C ");}
}
class MultipleInheritanceDemo
{
    public static void main(String arg[])
    {
        A a= new A();
        B b= new B();
    }
}

```

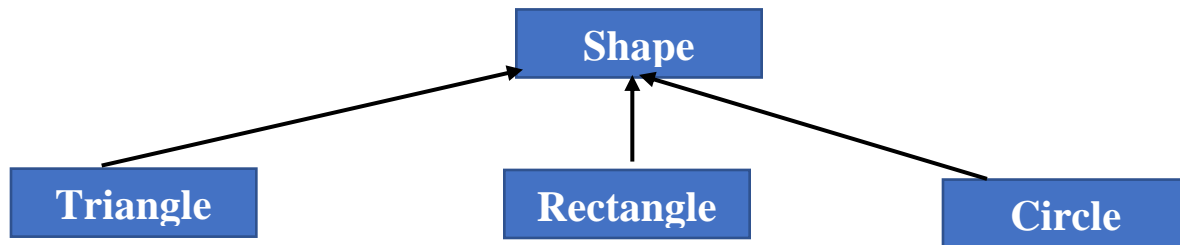
```

        C c= new C();
        c.method();
    }
}

```

// C.T. Error

Hierarchal Inheritance



class Shape

```

{
    void area() {}
}

```

class Triangle extends Shape

```

{
    int base,height;
    void getTriangleData(int b,int h)
    {
        base =b;
        height =h;
    }
    void area() {System.out.println("Triangle area="+0.5*base*height);}
}

```

class Rectangle extends Shape

```

{
    int length,width;
    void getRectangleData(int l,int w)
    {
        length =l;
        width =w;
    }
    void area() {System.out.println("Rectangle area="+length*width);}
}

```

class Circle extends Shape

```

{
    int radius ;
    void getCircleData(int r)
    {
        radius =r;
    }
    void area() {System.out.println("Circle area="+3.14*radius*radius);}
}

```

class HeirarichalInheritance

```

{

```

```

public static void main(String arg[])
{
    Triangle t= new Triangle();
    t.getTriangleData(3,4);
    Rectangle r= new Rectangle();
    r.getRectangleData(12,35);
    Circle c= new Circle();
    c.getCircleData(6);
    t.area();
    r.area();
    c.area();
}
}

```

/*Applying Method Overriding

Let's look at a more practical example that uses method overriding. The following program creates a superclass called Figure that stores the dimensions of various two-dimensional objects. It also defines a method called area() that computes the area of an object. The program derives two subclasses from Figure. The first is Rectangle and the second is Triangle. Each of these subclasses overrides area() so that it returns the area of a rectangle and a triangle, respectively.

Using run-time polymorphism.

```

class Figure {
    double dim1;
    double dim2;
    Figure(double a, double b)
    {
        dim1 = a;
        dim2 = b;
    }
    double area() {
        System.out.println("Area for Figure is undefined.");
        return 0;
    }
}

class Rectangle extends Figure {
    Rectangle (double a, double b) {
        super(a, b);
    }
}

```

```

    }
    // override area for rectangle
    double area() {
        System.out.println("Inside Area for Rectangle.");
        return dim1 * dim2;
    }
}

class Triangle extends Figure {
    Triangle(double a, double b) {
        super(a, b);
    }
    // override area for right triangle
    double area() {
        System.out.println("Inside Area for Triangle.");
        return dim1 * dim2 / 2;
    }
}

class FindAreasRTP {
    public static void main(String args[]) {
        Figure f = new Figure(10, 10);
        Rectangle r = new Rectangle(9, 5);
        Triangle t = new Triangle(10, 8);
        Figure figref;
        figref = r;
        System.out.println("Area is " + figref.area());
        figref = t;
        System.out.println("Area is " + figref.area());
        figref = f;
        System.out.println("Area is " + figref.area());
    }
}

```

- To access the data members of parent class when both parent and child class have member with same name.
 - To explicitly call the no-arg and parameterized constructor of parent class.
- 3) To access the method of parent class when child class has overridden that method.

Use super keyword to access the variables of parent class

When you have a variable in child class which is already present in the parent class then in order to access the variable of parent class, you need to use the super keyword.

Let's take an example to understand this: In the following program, we have a data member num declared in the child class, the member with the same name is already present in the parent class. There is no way you can access the num variable of parent class without using super keyword. .

```
class Superclass
{
    int num = 100;
}
class Subclass extends Superclass
{
    int num = 110;
    void printNumber(){
        System.out.println(num);    // super.num;
    }
    public static void main(String args[]){
        Subclass obj= new Subclass();
        obj.printNumber();
    }
}
```

Output: 110

Accessing the num variable of parent class

By calling a variable like this, we can access the variable of parent class if both the classes (parent and child) have same variable.

super.variable_name

Let's take the same example that we have seen above, this time in print statement we are passing super.num instead of num.

```
class Superclass
{
    int num = 100;
}
class Subclass extends Superclass
```



```

{
    int num = 110;
    void printNumber(){
        /* Note that instead of writing num we are
        * writing super.num in the print statement
        * this refers to the num variable of Superclass
        */
        System.out.println(super.num);
    }
    public static void main(String args[]){
        Subclass obj= new Subclass();
        obj.printNumber();
    }
}

```

Output: 100

As you can see by using **super.num** we accessed the num variable of parent class.

Use of super keyword to invoke constructor of parent class

When we create the object of sub class, the new keyword invokes the constructor of child class, which implicitly invokes the constructor of parent class. So the order to execution when we create the object of child class is: parent class constructor is executed first and then the child class constructor is executed. It happens because compiler itself adds `super()`(this invokes the no-arg constructor of parent class) as the first statement in the constructor of child class.

Let's see an example to understand what I have explained above:

class Parentclass

```

{
    Parentclass(){
        System.out.println("Constructor of parent class");
    }
}

```

class Subclass extends Parentclass

```

{
    Subclass(){
        /* Compile implicitly adds super() here as the

```

```

        * first statement of this constructor.
        */
        System.out.println("Constructor of child class");
    }
    Subclass(int num){
        /* Even though it is a parameterized constructor.
        * The compiler still adds the no-arg super() here
        */
        System.out.println("arg constructor of child class");
    }
    void display(){
        System.out.println("Hello!");
    }
    public static void main(String args[]){
        /* Creating object using default constructor. This
        * will invoke child class constructor, which will
        * invoke parent class constructor
        */
        Subclass obj= new Subclass();
        //Calling sub class method
        obj.display();
        /* Creating second object using arg constructor
        * it will invoke arg constructor of child class which will
        * invoke no-arg constructor of parent class automatically
        */
        Subclass obj2= new Subclass(10);
        obj2.display();
    }
}

```

Output:

```

Constructor of parent class
Constructor of child class
Hello!
Constructor of parent class
arg constructor of child class
Hello!

```

Parameterized super() call to invoke parameterized constructor of parent class

We can call super() explicitly in the constructor of child class, but it would not make any sense because it would be redundant. It's like explicitly doing something which would be implicitly done otherwise.

However when we have a constructor in parent class that takes arguments then we can use parameterized super, like `super(100);` to invoke parameterized constructor of parent class from the constructor of child class.

Let's see an example to understand this:

class Parentclass

```
{
    //no-arg constructor
    Parentclass(){
        System.out.println("no-arg constructor of parent class");
    }
    //arg or parameterized constructor
    Parentclass(String str){
        System.out.println("parameterized constructor of parent class");
    }
}
```

class Subclass extends Parentclass

```
{
    Subclass(){
        /* super() must be added to the first statement of constructor
        * otherwise you will get a compilation error. Another important
        * point to note is that when we explicitly use super in constructor
        * the compiler doesn't invoke the parent constructor automatically.
        */
        super("Hi");
        System.out.println("Constructor of child class");
    }
    void display(){
        System.out.println("Hello");
    }
    public static void main(String args[]){
        Subclass obj= new Subclass();
        obj.display();
    }
}
```

Output:

parameterized constructor of parent class
Constructor of child class
Hello

Note:

- super()(or parameterized super must be the first statement in constructor otherwise you will get the compilation error: “Constructor call must be the first statement in a constructor”
- When we explicitly placed super in the constructor, the java compiler didn't call the default no-arg constructor of parent class.

How to use super keyword in case of method overriding

When a child class declares a same method which is already present in the parent class then this is called method overriding. We will learn method overriding in the next tutorials of this series. For now you just need to remember this: When a child class overrides a method of parent class, then the call to the method from child class object always call the child class version of the method. However by using super keyword like this: super.method_name you can call the method of parent class (the method which is overridden). In case of method overriding, these terminologies are used: Overridden method: The method of parent class Overriding method: The method of child class Lets take an example to understand this concept:

class Parentclass

```
{
    //Overridden method
    void display(){
        System.out.println("Parent class method");
    }
}
class Subclass extends Parentclass
{
    //Overriding method
    void display(){
        System.out.println("Child class method");
    }
    void printMsg(){
        //This would call Overriding method
        display();
        //This would call Overridden method
        super.display();
    }
}
```

```

    }
    public static void main(String args[]){
        Subclass obj= new Subclass();
        obj.printMsg();
    }
}

```

Output:

Child class method

Parent class method

What if the child class is not overriding any method: No need of super

When child class doesn't override the parent class method then we don't need to use the super keyword to call the parent class method. This is because in this case we have only one version of each method and child class has access to the parent class methods so we can directly call the methods of parent class without using super.

class Parentclass

```

{
    void display(){
        System.out.println("Parent class method");
    }
}

```

class Subclass extends Parentclass

```

{
    void printMsg(){
        /* This would call method of parent class,
        * no need to use super keyword because no other
        * method with the same name is present in this class
        */
        display();
    }
    public static void main(String args[]){

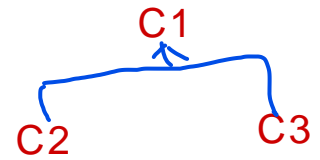
        Subclass obj= new Subclass();
        obj.printMsg();
    }
}

```

Output: Parent class method

Dynamic Method Dispatch(DMD)

```
class C1          // program To invoke super class constructor
{
    int i=20;
    void method() {System.out.println(" in C1"+i);}
}
class C2 extends C1
{
    void method()
    {
        System.out.println(" in C2");
    }
}
class C3 extends C1
{
    void method()
    {
        System.out.println(" in C3");
    }
}
class DMD1 {
    public static void main(String a[])
    { C1 obj1=new C1();
      C2 obj2=new C2();
      C3 obj3=new C3();
      C1 ref;
      ref=obj1;
      ref.method();
      ref=obj2;
      ref.method();
      ref=obj3;
      ref.method();
    }
}
```



class Bank

```
{  
    int getRateOfInterest(){return 0;} }  
    class SBI extends Bank{  
        int getRateOfInterest(){return 8;} }  
    class ICICI extends Bank{  
        int getRateOfInterest(){return 7;} }  
    class AXIS extends Bank{  
        int getRateOfInterest(){return 9;} }  
    class DMD3{  
public static void main(String args[]){  
    Bank bref;  
    SBI b1=new SBI();  
    bref=b1;  
    System.out.println("SBI Rate of Interest: "+bref.getRateOfInterest());  
    ICICI b2=new ICICI();  
    bref=b2;  
    System.out.println("ICICI Rate of Interest: "+bref.getRateOfInterest());  
    AXIS b3=new AXIS();  
    bref=b3;  
    System.out.println("AXIS Rate of Interest: "+bref.getRateOfInterest());  
}  
}
```