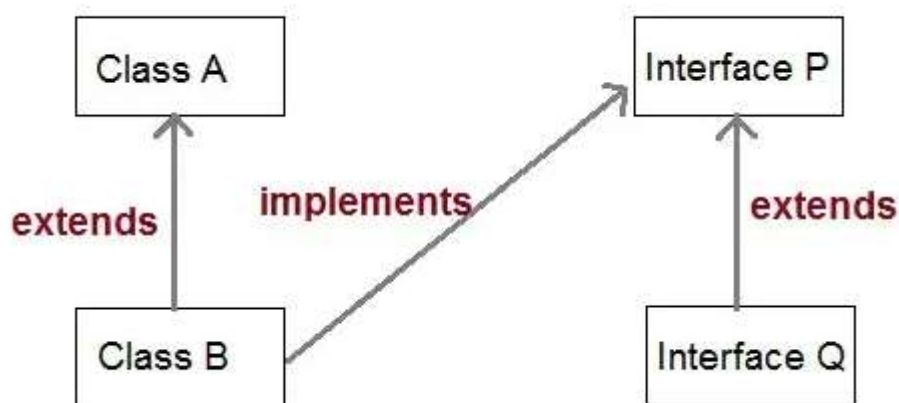


Inheritance

Inheritance is one of the key features of Object Oriented Programming. Inheritance provided mechanism that allowed a class to inherit properties of another class. When a Class extends another class it inherits all non-private members including fields and methods. Inheritance in Java can be best understood in terms of Parent and Child relationship, also known as Super class(Parent) and Sub class(child) in Java language.

Inheritance defines is-a relationship between a Super class and it's Sub class. extends and implements keywords are used to describe inheritance in Java.



Let us see how extends keyword is used to achieve Inheritance. It shows super class and sub-class relationship.

```
class Vehicle
{
    .....
}
class Car extends Vehicle
{
    ..... //extends the property of vehicle class
}
```

Now based on above example. In OOPs term we can say that,

Vehicle is super class of Car.

Car is sub class of Vehicle.

Car IS-A Vehicle.

Purpose of Inheritance

It promotes the code reusability i.e the same methods and variables which are defined in a parent/super/base class can be used in the child/sub/derived class.

It promotes polymorphism by allowing method overriding.

Disadvantages of Inheritance

Main disadvantage of using inheritance is that the two classes (parent and child class) gets tightly coupled.

This means that if we change code of parent class, it will affect to all the child classes which is inheriting/deriving the parent class, and hence, it cannot be independent of each other.

Simple example of Inheritance

```
class Parent
```

```
{
```

```
    void p1()
```

```
    {
```

```
        System.out.println("Parent method");
```

```
    }
```

```
}
```

```
class Child extends Parent {
```

```
    void c1()
```

```
    {
```

```
        System.out.println("Child method");
```

```
    }
```

```

    public static void main(String[] args)
    {
        Child cobj = new Child();
        cobj.c1(); //method of Child class
        cobj.p1(); //method of Parent class
    }
}

```

In the above code we have a class Parent which has a method p1(). Then we have created a new class Child which inherits the class Parent using the extends keyword and defines its own method c1(). Now by virtue of inheritance the class Child can also access the public method p1() of the class Parent.

Inheriting variables of super class

All the members of super class implicitly inherit to the child class. Member consists of instance variable and methods of the class.

Example

In this example the sub-class will be accessing the variable defined in the super class.

```

class Vehicle
{
    // variable defined
    String vehicleType;
}

class Car extends Vehicle {
    String modelType;
    void showDetail()
    {
        vehicleType = "Car"; //accessing Vehicle(Parent) class member
variable
        modelType = "Sports";
    }
}

```

```

        System.out.println(modelType + " " + vehicleType);
    }
    public static void main(String[] args)
    {
        Car car = new Car();
        car.showDetail();
    }
}

```

Types of Inheritance

Java mainly supports only three types of inheritance that are listed below.

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance

NOTE: Multiple inheritance is not supported in java

1. Single Inheritance

When a class extends to another class then it forms single inheritance. In the below example, we have two classes in which class A extends to class B that forms single inheritance.

Example

```

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

```

```

}
class B extends A
{
    public static void main(String[] args)
    {
        B b = new B();
        b.show();
    }
}

```

Here, we can notice that show() method is declared in class A, but using child class Demo object, we can call it. That shows the inheritance between these two classes.

2. Multilevel Inheritance

When a class extends to another class that also extends some other class forms a multilevel inheritance. For example, a class C extends to class B that also extends to class A and all the data members and methods of class A and B are now accessible in class C.

Example

```

class A{
    String a = "Class A";
    void showA() {
        System.out.println(a);
    }
}

class B extends A{
    String b = "Class B";
    void showB() {
        System.out.println(b);
    }
}

```

```

        }
    }
    class C extends B{
        public static void main(String[] args) {
            C c = new C();
            c.showA();
            c.showB();
        }
    }
}

```

3. Hierarchical Inheritance

When a class is extended by two or more classes, it forms hierarchical inheritance. For example, class B extends to class A and class C also extends to class A in that case both B and C share properties of class A.

Example

```

class A{
    String a = "Class A";
    void show() {
        System.out.println(a);
    }
}

class B extends A{
    String b = "Class B";
    void showB() {
        System.out.println(b);
    }
}

class C extends A{

```

```

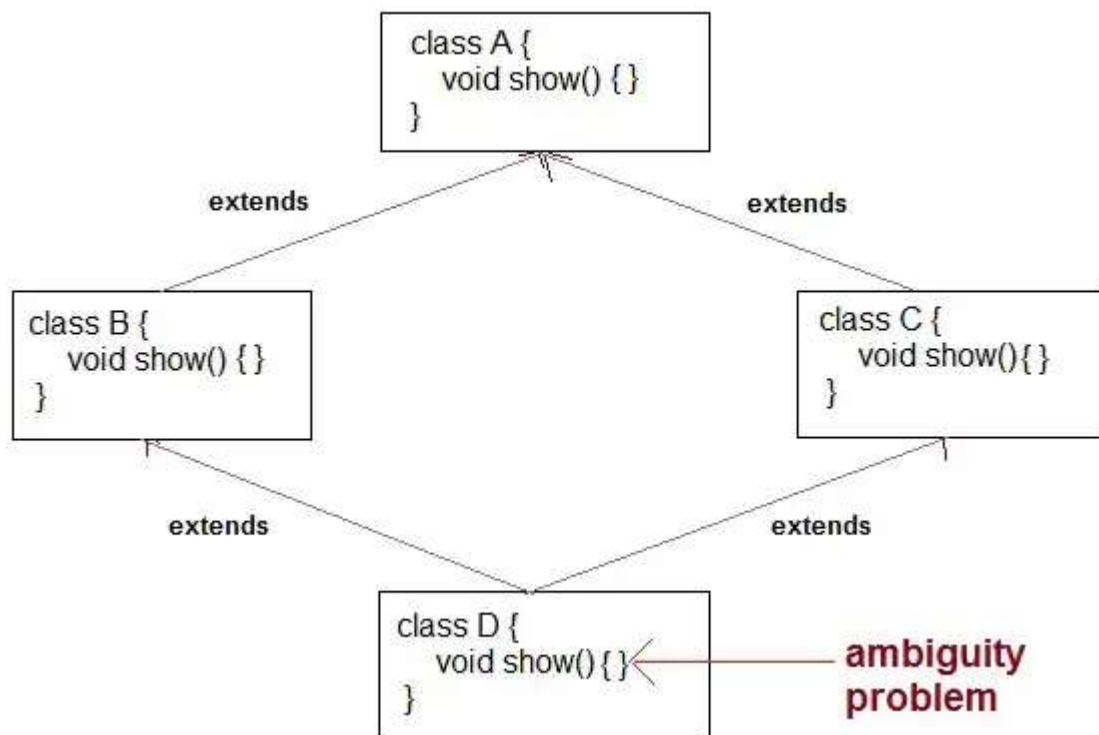
public static void main(String[] args) {
    C c = new C();
    c.show();
    B b = new B();
    b.show();
}
}

```

Why multiple inheritance is not supported in Java?

To remove ambiguity.

To provide more maintainable and clear design.

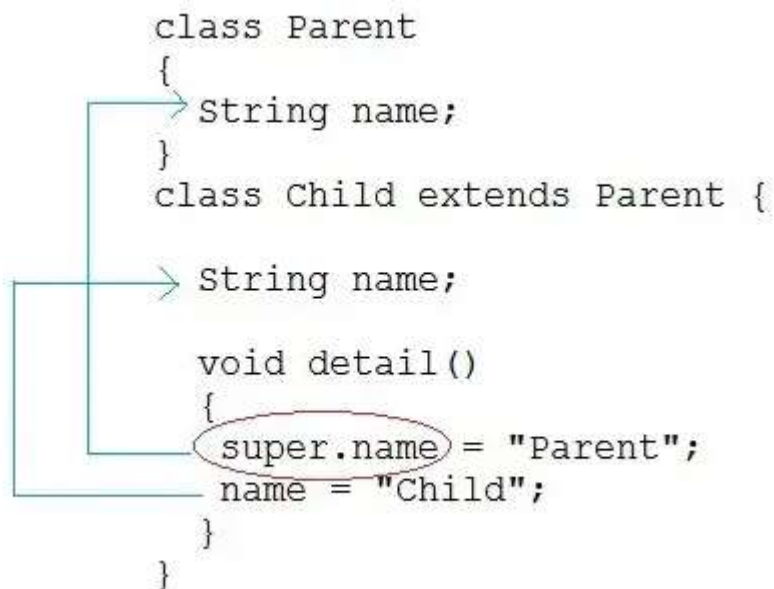


super keyword

In Java, super keyword is used to refer to immediate parent class of a child class. In other words, super keyword is used by a subclass whenever it needs to refer to its immediate super class.

```
class Parent
{
    String name;
}
class Child extends Parent {
    String name;

    void detail()
    {
        super.name = "Parent";
        name = "Child";
    }
}
```



Example of Child class referring Parent class property using super keyword

In this example we will only focus on accessing the parent class property or variables.

```
class Parent
{
    String name;
}

class Child extends Parent {
    String name;

    public void details()
    {
        super.name = "Parent"; //refers to parent class member
        name = "Child";
        System.out.println(super.name+" and "+name);
    }
}
```



```

    }
    public static void main(String[] args)
    {
        Child cobj = new Child();
        cobj.details();
    }
}

```

Example of Child class referring Parent class methods using super keyword

In this example we will only focus on accessing the parent class methods.

```

class Parent
{
    String name;
    public void details()
    {
        name = "Parent";
        System.out.println(name);
    }
}

class Child extends Parent {
    String name;
    public void details()
    {
        super.details();    //calling Parent class details() method
        name = "Child";
        System.out.println(name);
    }

    public static void main(String[] args)

```

```

    {
        Child cobj = new Child();
        cobj.details();
    }
}

```

Example of Child class calling Parent class constructor using super keyword

In this example we will focus on accessing the parent class constructor.

```
class Parent
```

```

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

```

```
class Child extends Parent {
```

```

    String name;
    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();  
}  
}
```

Note: When calling the parent class constructor from the child class using super keyword, super keyword should always be the first line in the method/constructor of the child class.