

Inheritance Review

Inheritance

It is the process where one class acquires the properties (methods and fields) of another. Information is made manageable in a hierarchical order.

Inheritance allows to 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. Moreover, new methods and fields can be added to a child class.



Is-a relationship

In Java, inheritance is an **is-a relationship**. That is, we use inheritance only if there exists an is-a relationship between two classes.

- Car is a Vehicle (Car can inherit from Vehicle)
- Orange is a Fruit (Orange can inherit from Fruit)
- Rose is a Flower (Rose can inherit from Flower)
- Dog is an Animal (Dog can inherit from Animal)



Superclass and Subclass

Subclass/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

Superclass/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.



Inheritance - Main advantage

Reusability: It is a mechanism which facilitates to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.



Java Inheritance Syntax

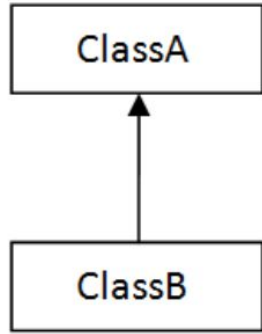
The **extends** keyword indicates that you are making a new class that derives from an existing class.

class Subclass-name **extends** Superclass-name

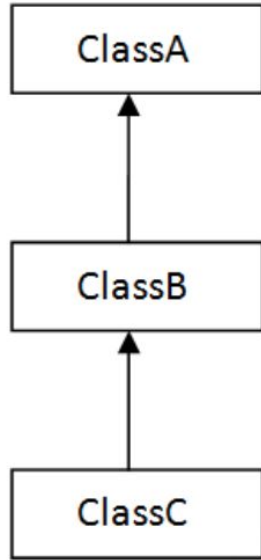
```
{  
    //methods and fields  
}
```



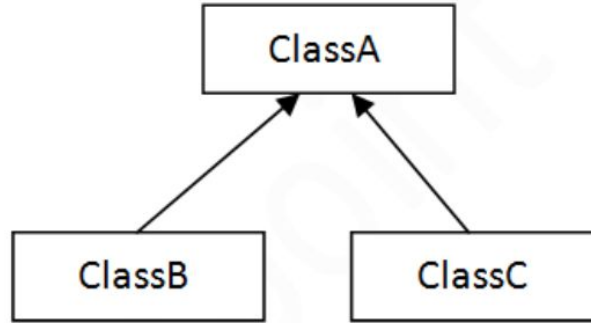
Types of inheritance in Java



1) Single



2) Multilevel



3) Hierarchical

Single inheritance

```
class Animal{
    public void eat(){
        System.out.println("eating...");
    }
}

class Dog extends Animal{
    public void bark(){
        System.out.println("barking...");
    }
}

public class TestInheritance{
    public static void main(String args[]){
        Dog d=new Dog();
        d.bark();
        d.eat();
    }
}
```

When a class inherits another class.



Multilevel Inheritance

```
class Animal{
    public void eat(){
        System.out.println("eating...");
    }
}

class Dog extends Animal{
    public void bark(){
        System.out.println("barking...");
    }
}

class BabyDog extends Dog{
    public void weep(){
        System.out.println("weeping...");
    }
}

public class TestInheritance{
    public static void main(String args[]){
        BabyDog d=new BabyDog();
        d.weep();
        d.bark();
        d.eat();
    }
}
```

When there is a chain of inheritance.



Hierarchical Inheritance

```
class Animal{
    public void eat(){
        System.out.println("eating...");
    }
}

class Dog extends Animal{
    public void bark(){
        System.out.println("barking...");
    }
}

class Cat extends Animal{
    public void meow(){
        System.out.println("meowing...");
    }
}

public class TestInheritance{
    public static void main(String args[]){
        Dog d = new Dog();
        d.bark();
        d.eat();
        Cat c = new Cat();
        c.meow();
        c.eat();
    }
}
```

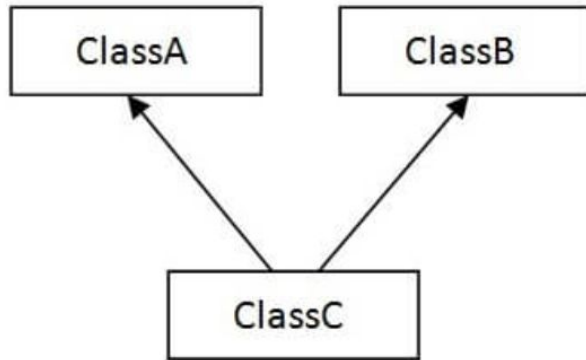
When two or more classes inherits a single class.



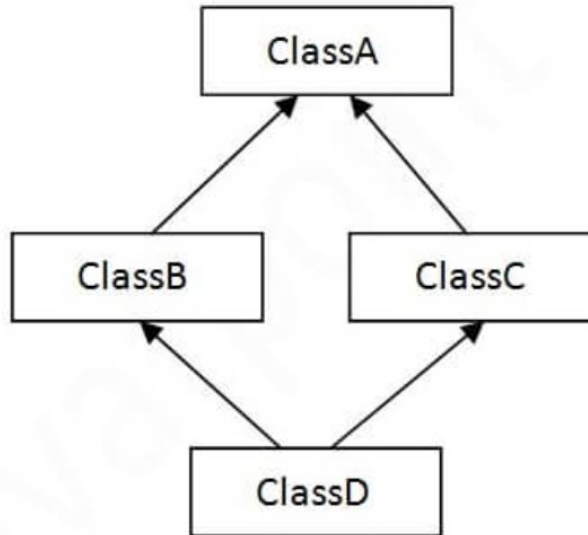
Multiple inheritance is not supported in Java

When one class inherits multiple classes, it is known as multiple inheritance.

To reduce complexity and simplify the language, multiple inheritance is not supported in Java.



4) Multiple



5) Hybrid

What does it happen when the same method is defined in both the superclass and the subclass?



Method Overriding

If the same method is defined in both the superclass and the subclass, then the method of the subclass class overrides the method of the superclass. This is known as **method overriding**.

```
class Animal {  
    public void displayInfo() {  
        System.out.println("I am an animal.");  
    }  
}  
  
class Dog extends Animal {  
    @Override  
    public void displayInfo() {  
        System.out.println("I am a dog.");  
    }  
}  
  
class Main {  
    public static void main(String[] args) {  
        Dog d1 = new Dog();  
        d1.displayInfo();  
    }  
}
```

`@Override` is an annotation. Java annotations are used to provide information to the compiler. The `@Override` annotation specifies the compiler that the method after this annotation overrides the method of the superclass.

It is not mandatory to use `@Override`.

Rules: Both the superclass and the subclass must have the same method name, the same return type and the same parameter list.

super Keyword in Java Overriding

To access the members (attributes, constructors and methods) of the superclass from the subclass, we use the **super keyword**.

```
class Animal {  
    public void displayInfo() {  
        System.out.println("I am an animal.");  
    }  
}  
  
class Dog extends Animal {  
    public void displayInfo() {  
        super.displayInfo();  
        System.out.println("I am a dog.");  
    }  
}  
  
class Main {  
    public static void main(String[] args) {  
        Dog d1 = new Dog();  
        d1.displayInfo();  
    }  
}
```

What is the output of this code?



Uses of super keyword

- To call methods of the superclass that is overridden in the subclass.
 - To access attributes (fields) of the superclass if both superclass and subclass have attributes with the same name.
1. To explicitly call superclass no-arg (default) or parameterized constructor from the subclass constructor.



Constructors

Constructors in Java are not inherited. Thus, there is not a constructor overriding in Java.

A constructor of the superclass can be called from its subclasses. For that, we use `super()`



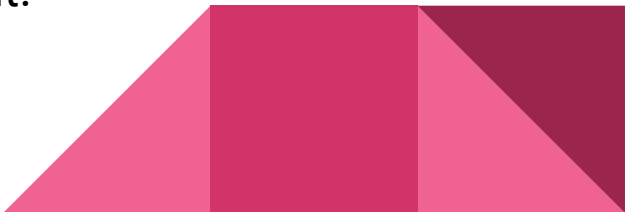
Use of super() to access superclass constructor

```
class Animal {  
    // default or no-arg constructor of class Animal  
    Animal() {  
        System.out.println("I am an animal");  
    }  
}  
  
class Dog extends Animal {  
    // default or no-arg constructor of class Dog  
    Dog() {  
        // calling default constructor of the superclass  
        super();  
        System.out.println("I am a dog");  
    }  
}  
  
class Main {  
    public static void main(String[] args) {  
        Dog dog1 = new Dog();  
    }  
}
```

When an object of a class is created, its default constructor is automatically called.

To explicitly call the superclass constructor from the subclass constructor, we use `super()`.

`super()` can be used only inside the subclass constructor and must be the first statement.



Call Parameterized Constructor Using super()

```
// default or no-arg constructor
Animal() {
    System.out.println("I am an animal");
}

// parameterized constructor
Animal(String type) {
    System.out.println("Type: "+type);
}
}

class Dog extends Animal {

    // default constructor
    Dog() {

        // calling parameterized constructor of the superclass
        super("Animal");

        System.out.println("I am a dog");
    }
}

class Main {
    public static void main(String[] args) {
        Dog dog1 = new Dog();
    }
}
```

Superclass references

A **superclass reference variable** can hold an object of that superclass or of any of its subclasses.

```
Animal firstAnimal = new Animal();  
Animal secondAnimal = new Dog();
```

The opposite is NOT true. **You CANNOT declare a variable of the subclass to reference a superclass object.**

```
Dog myDog = new Animal(); // ERROR
```



Practice

- Work with a partner and think on an inheritance example to implement
- Requirements:
 - One superclass and at least two subclasses
 - Implement a default constructor and a constructor with parameters in each class
 - You must override at least one method in each subclass
- Two or three groups will present their work to the class

