Welcome you all

JAVA PROGRAMMING

DAY : 5

D.Sakthivel
Assistant Professor & Trainer,
KGiSL Micro College
KGiSL Campus, Coimbatore – 641 035.

Day 5

- Inheritance
- Super Keyword
- Types of Inheritance

Inheritance in Java

- **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.
- Moreover, you can add new methods and fields in your current class also.
- Inheritance represents the ***parent-child relationship***.

Why use inheritance in java

1. For Method Overriding (so runtime polymorphism can be achieved).
2. For Code Reusability.

Terms used in Inheritance

Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

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

Terms used in Inheritance

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

Reusability: As the name specifies, reusability is a mechanism which facilitates you 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.

The syntax of Java Inheritance

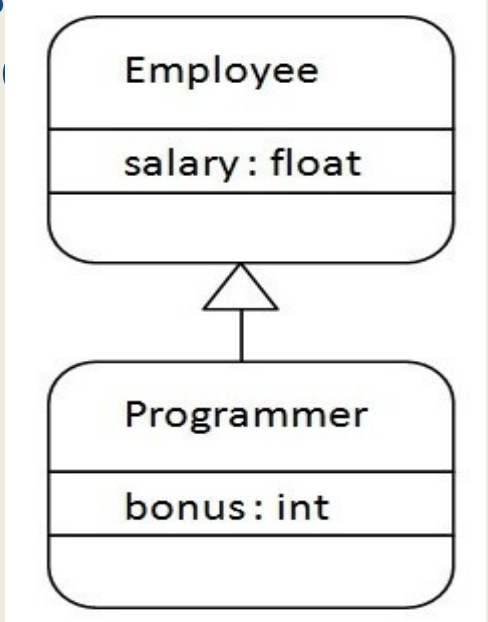
```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

Inheritance

- 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.
- In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

Inheritance

- 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.
- In the terminology of Java, a class which is inherited is called a parent or superclass and the new class is called chil



KGiSL MicroCollege

Inheritance

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.sal
ary);
   System.out.println("Bonus of Programmer is:"+p.
bonus);
}
}
```

```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

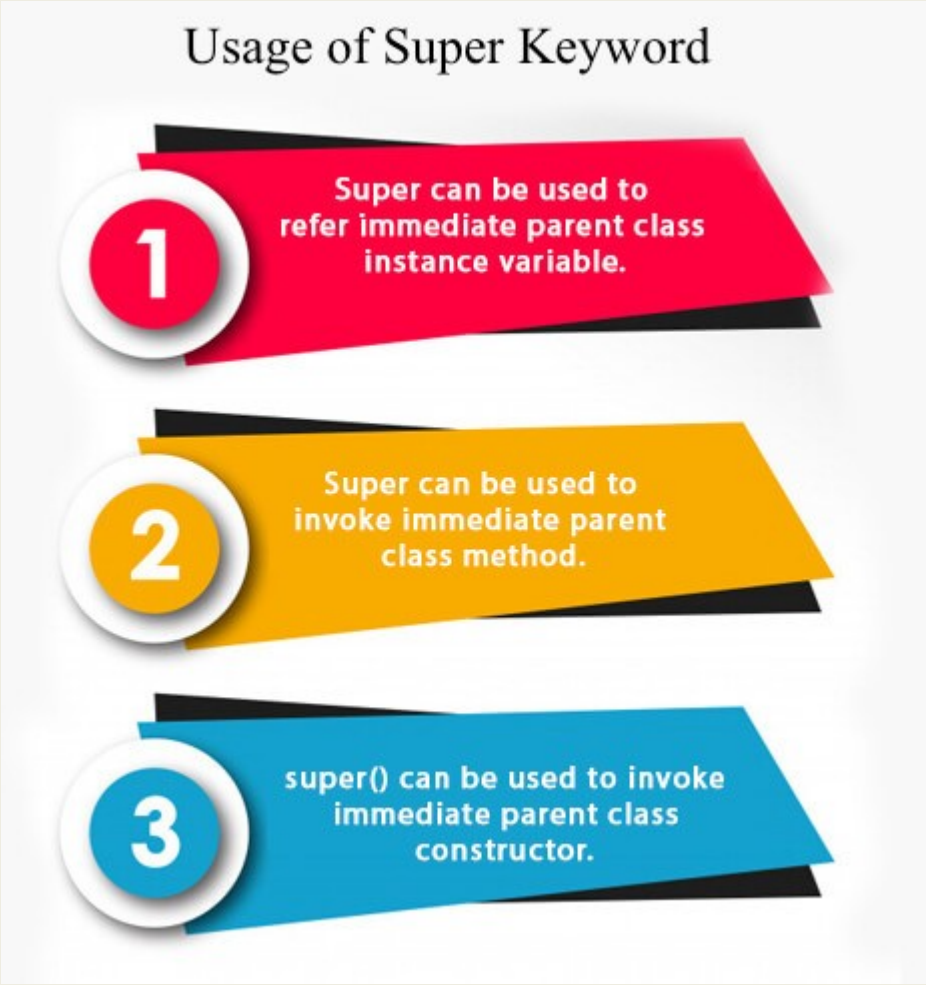
The super keyword

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 immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate ructor.



KGiSL MicroCollege

super is used to refer immediate parent class instance variable.

```
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 TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}}
```

super is used to refer immediate parent class instance variable.

```
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 TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}}
```

Output:

```
black
white
```

KGiSL MicroCollege

super can be used to invoke parent class method.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog 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[]){
Dog d=new Dog();
d.work();
}}
```

Output:

```
eating...
barking...
```

KGiSL MicroCollege

super is used to invoke parent class constructor.

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

Output:

```
animal is created
dog is created
```

KGiSL MicroCollege

Real Use.

```
class Person{
int id;
String name;
Person(int id,String name){
this.id=id;
this.name=name;
}
}
class Emp extends Person{
float salary;
Emp(int id,String name,float salary){
super(id,name);//reusing parent constructor
this.salary=salary;
}
void display()
{System.out.println(id+" "+name+" "+salary);}
}
class TestSuper5{
public static void main(String[] args){
Emp e1=new Emp(1,"ankit",45000f);
e1.display();
}}
```

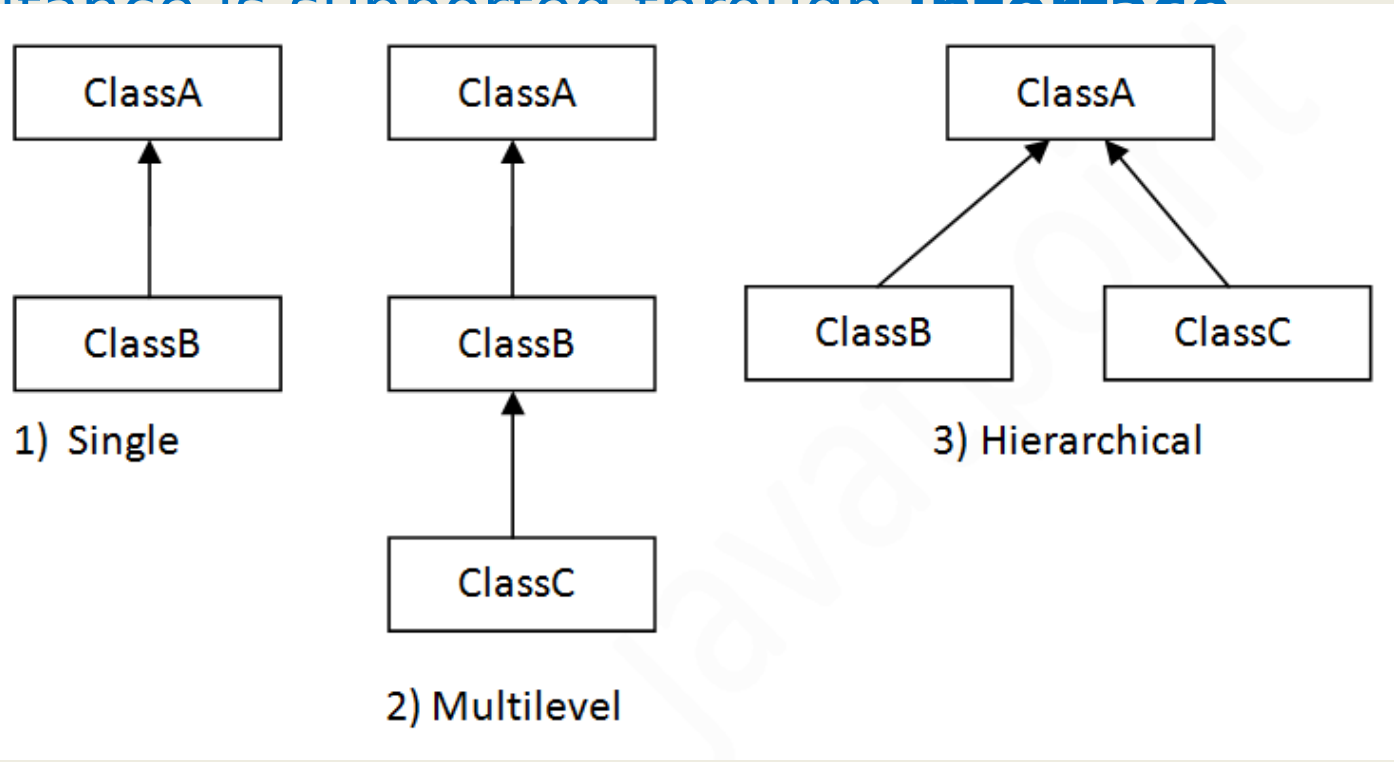
Output:

```
1 ankit 45000
```

KGiSL MicroCollege

Types of inheritance in java

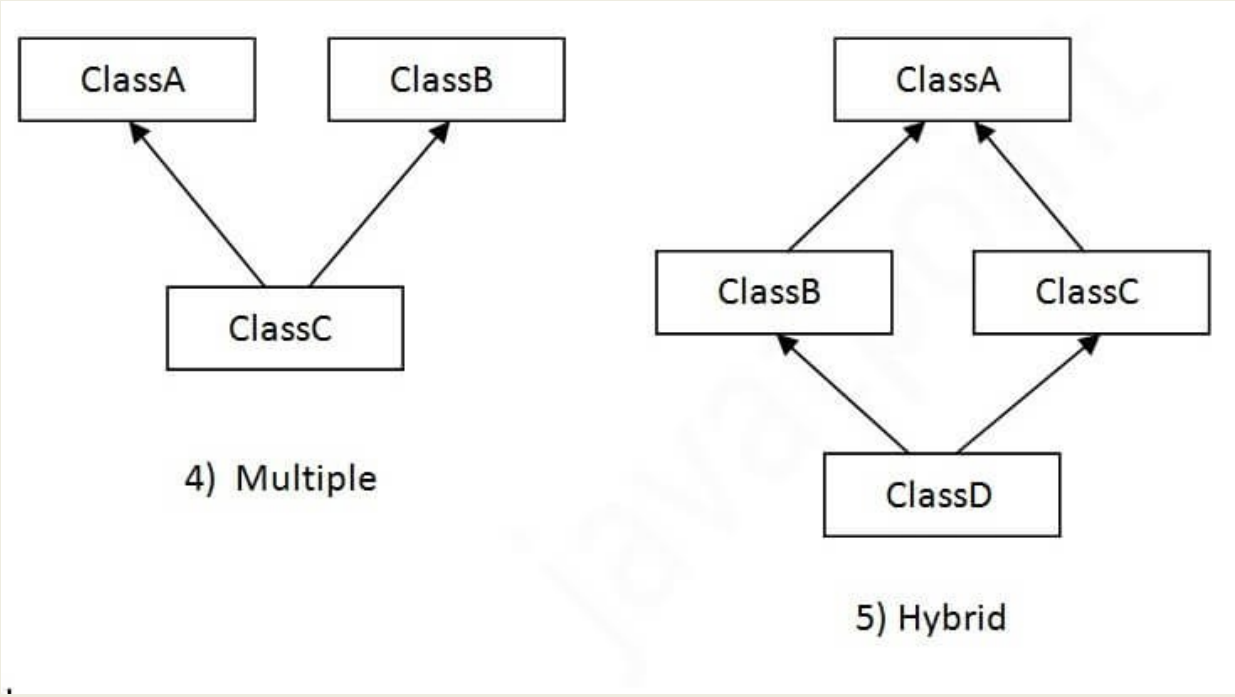
- On the basis of class, there can be three types of inheritance in java: **single, multilevel and hierarchical.**
- In java programming, multiple and hybrid inheritance is supported through **interface only**



Types of inheritance in java

Note: Multiple inheritance is not supported in Java through class.

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



Single Inheritance

When a class inherits another class, it is known as a ***single inheritance*****.** In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

File: TestInheritance.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

Output:

```
barking...
eating...
```

KGiSL MicrCollege

Multilevel Inheritance

When there is a chain of inheritance, it is known as ***multilevel inheritance*.** As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark()
{System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep()
{System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Output:

```
weeping...
barking...
eating...
```

KGiSL MicrCollege

Multilevel Inheritance

When there is a chain of inheritance, it is known as ***multilevel inheritance*.** As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark()
{System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep()
{System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Output:

```
weeping...
barking...
eating...
```

KGiSL MicrCollege

Hierarchical Inheritance

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

In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Output:

```
meowing...
eating...
```

Why multiple inheritance is not supported in java?

- To reduce the complexity and simplify the language, multiple inheritance is not supported in java.
- Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.
- Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be pile time error.

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 public static void main(String args[]){
   C obj=new C();
   obj.msg();//
Now which msg() method would be invoked?
}
}
```

```
Compile Time Error
```

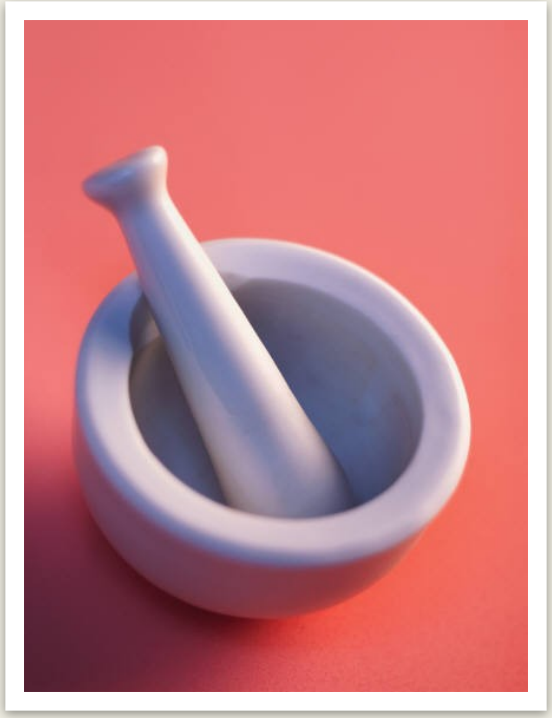
Q) Why multiple inheritance is not supported in java?

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

- Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes.
- If A and B classes have the same method and you call it from child class object, there will be **ambiguity to call the method of A or B class.**
- Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes.

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 public static void main(String args[]){
   C obj=new C();
   obj.msg();//
Now which msg() method would be invoked?
}
}
```



Thank You