**METHOD OVERLOADING IN JAVA**

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

**Advantage of method overloading**

1. Method overloading increases the readability of the program.

**There are two ways to overload the method in java**

1. By changing number of arguments
2. By changing the data type

**Method Overloading: changing no. of arguments**

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

class Adder{

static int add(int a,int b){return a+b;}

static int add(int a,int b,int c){return a+b+c;}

}

class TestOverloading1{

public static void main(String[] args){

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(11,11,11));

}}

**Method Overloading: changing data type of arguments**

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

class Adder{

static int add(int a, int b){return a+b;}

static double add(double a, double b){return a+b;}

}

class TestOverloading2{

public static void main(String[] args){

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(12.3,12.6));

}}

**Why Method Overloading is not possible by changing the return type of method only?**

In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:

class Adder{

static int add(int a,int b){return a+b;}

static double add(int a,int b){return a+b;}

}

class TestOverloading3{

public static void main(String[] args){

System.out.println(Adder.add(11,11));//ambiguity

}}

**Can we overload java main() method?**

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only. Let's see the simple example:

class TestOverloading4{

public static void main(String[] args){System.out.println("main with String[]");}

public static void main(String args){System.out.println("main with String");}

public static void main(){System.out.println("main without args");}

}

**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.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

**Usage of Java Method Overriding**

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

**Rules for Java Method Overriding**

The method must have the same name as in the parent class

The method must have the same parameter as in the parent class.

There must be an IS-A relationship (inheritance).

**Understanding the problem without method overriding**

Let's understand the problem that we may face in the program if we don't use method overriding.

class Vehicle{

void run(){System.out.println("Vehicle is running");}

}

//Creating a child class

class Bike extends Vehicle{

public static void main(String args[]){

//creating an instance of child class

Bike obj = new Bike();

//calling the method with child class instance

obj.run();

}

}

Problem is that I have to provide a specific implementation of run() method in subclass that is why we use method overriding.

**Example of method overriding**

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method are the same, and there is IS-A relationship between the classes, so there is method overriding.

class Vehicle{

//defining a method

void run(){System.out.println("Vehicle is running");}

}

//Creating a child class

class Bike2 extends Vehicle{

//defining the same method as in the parent class

void run(){System.out.println("Bike is running safely");}

public static void main(String args[]){

Bike2 obj = new Bike2();//creating object

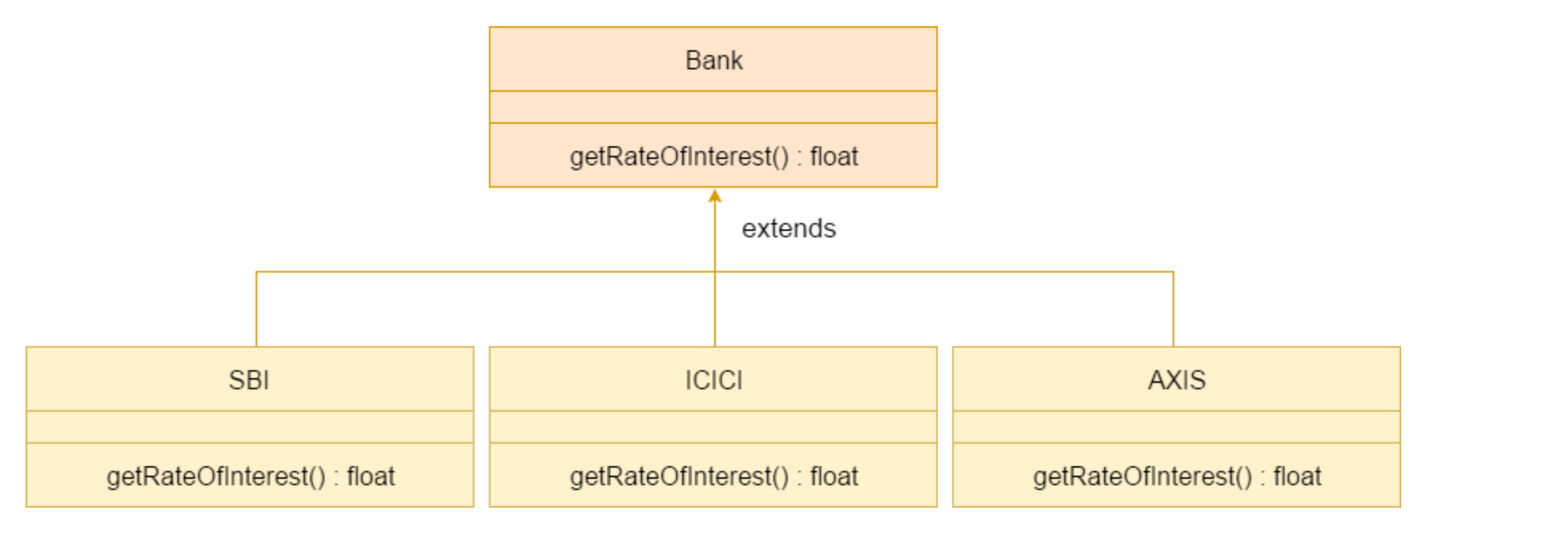
obj.run();//calling method

}

}

**A real example of Java Method Overriding**

Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest.



class Bank{

int getRateOfInterest(){return 0;}

}

//Creating child classes.

class SBI extends Bank{

int getRateOfInterest(){return 8;}

}

class ICICI extends Bank{

int getRateOfInterest(){return 7;}

}

class AXIS extends Bank{

int getRateOfInterest(){return 9;}

}

//Test class to create objects and call the methods

class Test2{

public static void main(String args[]){

SBI s=new SBI();

ICICI i=new ICICI();

AXIS a=new AXIS();

System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());

System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());

System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());

}

}

**SUPER KEYWORD IN JAVA**

The super keyword in Java is a reference variable which is used to refer immediate parent class object.

**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 parent class constructor.

**super 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 TestSuper1{

public static void main(String args[]){

Dog d=new Dog();

d.printColor();

}}

**super can be used to invoke parent class method**

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

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();

}}

**super is used to invoke parent class constructor.**

The super keyword can also be used to invoke the parent class constructor. Let's see a simple example:

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();

}}

**FINAL KEYWORD IN JAVA**

The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called **blank final variable** or **uninitialized final variable**. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

**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 Bike9{

final int speedlimit=90;//final variable

void run(){

speedlimit=400;

}

public static void main(String args[]){

Bike9 obj=new Bike9();

obj.run();

}

}//end of class

**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();

}

}

**Java final class**

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

final class Bike{}

class Honda1 extends Bike{

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

public static void main(String args[]){

Honda1 honda= new Honda1();

honda.run();

}

}

**Is final method inherited?**

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

class Bike{

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

}

class Honda2 extends Bike{

public static void main(String args[]){

new Honda2().run();

}

}