SNS COLLEGE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION)

COIMBATORE – 35

DEPARTMENT OF COMPUTER SIENCE AND ENGINEERING

UNIT III – INHERITANCE AND POLYMORPHISM

Inheritance
Super class, Sub class
Types of inheritance
Method Overloading
Method Overriding
Abstract class
this keyword, Final keyword
Packages
Interfaces

Method Overriding in Java

- Method Overriding is a type of **runtime polymorphism**.
- In method overriding, a method in a derived class has the same name, return type, and parameters as a method in its parent class.
- The derived class provides a specific implementation for the method that is already defined in the parent class.

Example 1 of Method Overriding:

```
import java.io.*;

// Base Class
class Animal
{
    void eat()
    {
        System.out.println("eat() method of base class");
        System.out.println("Animal is eating.");
    }
}

// Derived Class
class Dog extends Animal
{
    @Override
    void eat()
    {
        System.out.println("eat() method of derived class");
        System.out.println("Dog is eating.");
    }

    // Method to call the base class method
    void eatAsAnimal()
    {
        super.eat();
    }
}

// Driver Class
class MethodOverridingEx
{
    // Main Function
    public static void main(String args[])
     {
        Dog d1 = new Dog();
        Animal a1 = new Animal();

        d1.eat();              // Calls the eat() method of Dog class
        a1.eat();              // Calls the eat() method of Animal class


        // Polymorphism: Animal reference pointing to Dog object
   Animal animal = new Dog();

    animal.eat();              // Calls the eat() method of Dog class

((Dog) animal).eatAsAnimal(); // To call the base class method, you need to use a Dog
reference

    }
}
```

Output

```
eat() method of derived class
Dog is eating.

eat() method of base class
Animal is eating.

eat() method of derived class
Dog is eating.

eat() method of base class
Animal is eating.
```

Explanation of the above Program:

Here, we can see that a method eat() has overridden in the derived class name **Dog** that is already provided by the base class name **Animal**. When we create the instance of class Dog and call the eat() method, we see that only derived class eat() method run instead of base class method eat(), and When we create the instance of class Animal and call the eat() method, we see that only base class eat() method run instead of derived class method eat().

Example 2 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.

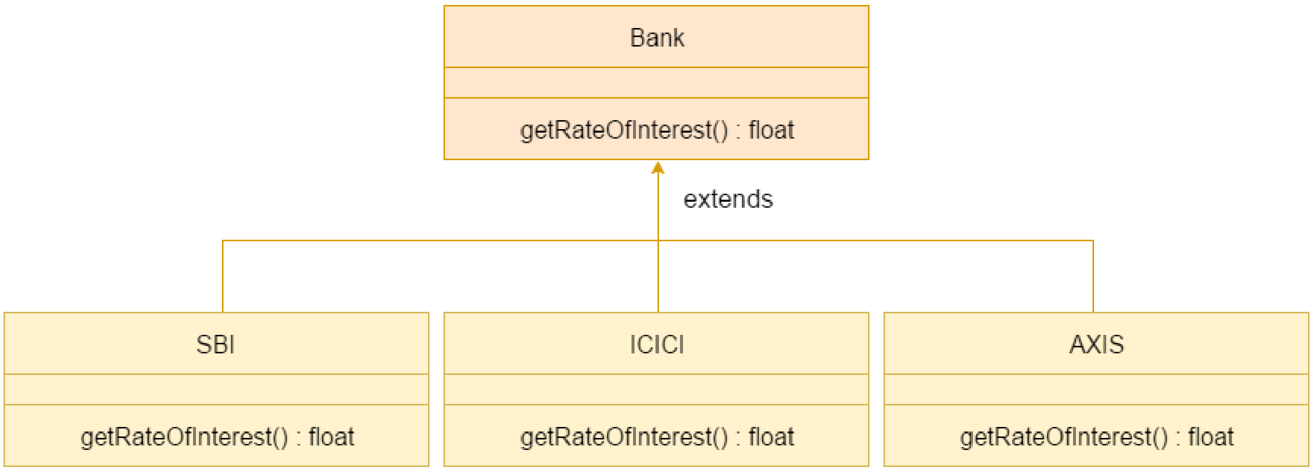
```
//Java Program to illustrate the use of Java Method Overriding
//Creating a parent class.
class Vehicle
{
  //defining a method
  void run()
     {System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike2 extends Vehicle
{
  //defining the same method as in the parent class
  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
  }
}
```

Output:

```
Bike is running safely
```

Example 3 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.


Bank
getRateOfInterest() : float
extends
SBI
getRateOfInterest() : float
ICICI
getRateOfInterest() : float
AXIS
getRateOfInterest() : float

Java method overriding is mostly used in Runtime Polymorphism which we will learn in next pages.

```
//Java Program to demonstrate the real scenario of Java Method Overriding
//where three classes are overriding the method of a parent class.

//Creating a parent class.
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());
}
}
```

```
Output:
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

Can we override static method?

No, a static method cannot be overridden. It can be proved by runtime polymorphism, so we will learn it later.

Why can we not override static method?

It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.

Can we override java main method?

No, because the main is a static method.

The differences between Method Overloading and Method Overriding in Java are as follows:

Method Overloading	**Method Overriding**
Method overloading is a compile-time polymorphism.	Method overriding is a run-time polymorphism.
Method overloading helps to increase the readability of the program.	Method overriding is used to grant the specific implementation of the method which is already provided by its parent class or superclass.
It occurs within the class.	It is performed in two classes with inheritance relationships.
Method overloading may or may not require inheritance.	Method overriding always needs inheritance.
In method overloading, methods must have the same name and different signatures.	In method overriding, methods must have the same name and same signature.
In method overloading, the return type can or can not be the same, but we just have to change the parameter.	In method overriding, the return type must be the same or co-variant.
Static binding is being used for overloaded methods.	Dynamic binding is being used for overriding methods.
Private and final methods can be overloaded.	Private and final methods can’t be overridden.
The argument list should be different while doing method overloading.	The argument list should be the same in method overriding.