Method Overriding in Java

Understanding the problem without method overriding

Can we override the static method

Method overloading vs. method overriding

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).

Java Rules for Method Overriding

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.

//Java Program to demonstrate why we need method overriding

//Here, we are calling the method of parent class with child

//class object.

//Creating a parent class

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();

}

}

Test it Now

Output:

Vehicle is running

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.

//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

}

}

Test it Now

Output:

Bike is running safely

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.



Java method overriding example of bank

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());

}

}

Test it Now

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.

Difference between method Overloading and Method Overriding in java

Click me for the difference between method overloading and overriding

More topics on Method Overriding (Not For Beginners)

Method Overriding with Access Modifier

Let's see the concept of method overriding with access modifier.

Exception Handling with Method Overriding

Let's see the concept of method overriding with exception handling.

Covariant Return Type

The covariant return type specifies that the return type may vary in the same direction as the subclass.

Before Java5, it was not possible to override any method by changing the return type. But now, since Java5, it is possible to override method by changing the return type if subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type. Let's take a simple example:

Note: If you are beginner to java, skip this topic and return to it after OOPs concepts.

Simple example of Covariant Return Type

class A{

A get(){return this;}

}

class B1 extends A{

B1 get(){return this;}

void message(){System.out.println("welcome to covariant return type");}

public static void main(String args[]){

new B1().get().message();

}

}

Test it Now

Output:welcome to covariant return type

As you can see in the above example, the return type of the get() method of A class is A but the return type of the get() method of B class is B. Both methods have different return type but it is method overriding. This is known as covariant return type.

How is Covariant return types implemented?

Java doesn't allow the return type based overloading but JVM always allows return type based overloading. JVM uses full signature of a method for lookup/resolution. Full signature means it includes return type in addition to argument types. i.e., a class can have two or more methods differing only by return type. javac uses this fact to implement covariant return types.