Difference Between Method Overloading and Method Overriding in Java

Difficulty Level : Medium Last Updated : 02 Feb, 2022

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

S.NO	Method Overloading	Method Overriding
1.	Method overloading is a compile-time polymorphism.	Method overriding is a run-time polymorphism.
2.	It helps to increase the readability of the program.	It is used to grant the specific implementation of the method which is already provided by its parent class or superclass.
3.	It occurs within the class.	It is performed in two classes with inheritance relationships.
4.	Method overloading may or may not require inheritance.	Method overriding always needs inheritance.
5.	In method overloading, methods must have the same name and different signatures.	In method overriding, methods must have the same name and same signature.
6.	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.

Method Overloading:

Method Overloading is a **Compile time polymorphism**. In method overloading, more than one method shares the same method name with a different signature in the class. In method overloading, the return type can or can not be the same, but we have to change the parameter because, in java, we can not achieve the method overloading by changing only the return type of the method.

Example of Method Overloading:

```
import java.io.*;

class MethodOverloadingEx {

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

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

    public static void main(String args[])
    {
          System.out.println("add() with 2 parameters");
        System.out.println(add(4, 6));

          System.out.println("add() with 3 parameters");
        System.out.println(add(4, 6, 7));
    }
}
```

Output

```
add() with 2 parameters
10
add() with 3 parameters
17
```

Method Overriding:

Method Overriding is a **Run time polymorphism**. In method overriding, the derived class provides the specific implementation of the method that is already provided by the base class or parent class. In method overriding, the return type must be the same or co-variant (return type may vary in the same direction as the derived class).

Example of Method Overriding:

```
import java.io.*;

class Animal {

    void eat()
    {
        System.out.println("eat() method of base class");
        System.out.println("eating.");
    }
}

class Dog extends Animal {

    void eat()
    {
        System.out.println("eat() method of derived class");
        System.out.println("Dog is eating.");
    }
}

class MethodOverridingEx {

    public static void main(String args[])
    {
        Dog d1 = new Dog();
        Animal a1 = new Animal();

        d1.eat();
        a1.eat();

        Animal animal = new Dog();
        // eat() method of animal class is overridden by
        // base class eat()
        animal.eat();
    }
}
```

Output

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

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

Explanation:

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().

So, it’s clear that in method overriding, the method is bound to the instances on the run time, which is decided by the **JVM**. That’s why it is called **Run time polymorphism.**