Method Overriding in Java

Defining the method again with the same name and parameters but in different classes is called method Overriding. If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding**.

In other words, if a subclass provides the specific implementation of the method that has been provided by one of its parent classes, it is known as Method Overriding. Also known as Runtime Polymorphism/Dynamic polymorphism/Late binding.
Overriding method resolution is also known as "Dynamic Method Dispatch"

ADVANTAGE OF JAVA METHOD OVERRIDING

- Method Overriding is used to provide a specific implementation of a method that is already provided by its super class.
- Method Overriding is used for Runtime Polymorphism.

RULES FOR METHOD OVERRIDING

1. The method must have the same name as in the parent class.
2. The method must have the same parameter as in the parent class.
3. 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.

```
1.  class Vehicle{
2.  void run()
3.  {
4.     System.out.println("Vehicle is running");}
5.  }
6.  class Bike extends Vehicle
7. {
8.  public static void main(String args[]){
9.  Bike obj = new Bike();
10. obj.run();
11. }
12. }
```

Output: The vehicle is running

The problem is that I have to provide a specific implementation of the run() method in a subclass which is why we use 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 an IS-A relationship between the classes, so there is method overriding.

```
1. class Vehicle{
2. void run(){System.out.println("Vehicle is running");}
3. }

4. class Bike extends Vehicle{
5. void run(){System.out.println("Bike is running safely");
6. }
7.
8. public static void main(String args[]){
9. Bike obj = new Bike();
10. obj.run();
11. }
```

Output: Bike is running safely

A REAL EXAMPLE OF A JAVA METHOD OVERRIDING

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

```
1. class Bank{
2. int getRateOfInterest(){return 0;}
3. }
4.
5. class SBI extends Bank{
6. int getRateOfInterest(){return 8;}
7. }
8.
```

```
9.  class ICICI extends Bank{
10. int getRateOfInterest(){return 7;}
11. }
12. class AXIS extends Bank{
13. int getRateOfInterest(){return 9;}
14. }
15.
16. class Test{
17. public static void main(String args[]){
18. SBI s=new SBI();
19. ICICI i=new ICICI();
20. AXIS a=new AXIS();
21. System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
22. System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
23. System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
24. }
25. }
```

Output:

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

instanceof operator

The **java instanceof operator** is used to test whether the object is an instance of the specified type (class or subclass or interface).

The instanceof in Java is also known as type *comparison operator* because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

```
1  class Animal {
2  }
3
4  class Mammal extends Animal {
5  }
6
7  class Reptile extends Animal {
8  }
9
10 public class Dog extends Mammal {
11
12    public static void main(String args[]) {
13        Animal a = new Animal();
14        Mammal m = new Mammal();
15        Dog d = new Dog();
16
17        System.out.println(m instanceof Animal);
18        System.out.println(d instanceof Mammal);
19        System.out.println(d instanceof Animal);
20    }
21 }
```

OUTPUT: