**Method Overriding in Java**

1. [Understanding problem without method overriding](http://www.javatpoint.com/method-overriding-in-java#moverproblem)
2. [Can we override the static method](http://www.javatpoint.com/method-overriding-in-java#movercanstatic)
3. [method overloading vs method overriding](http://www.javatpoint.com/method-overriding-in-java#moverdiff)

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding**.

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

**Advantage of Java Method Overriding**

* Method Overriding is used to provide 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. method must have same name as in the parent class
2. method must have same parameter as in the parent class.
3. must be 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(){System.out.println("Vehicle is running");}
3. }
4. class Bike extends Vehicle{
6. public static void main(String args[]){
7. Bike obj = new Bike();
8. obj.run();
9. }
10. }

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**

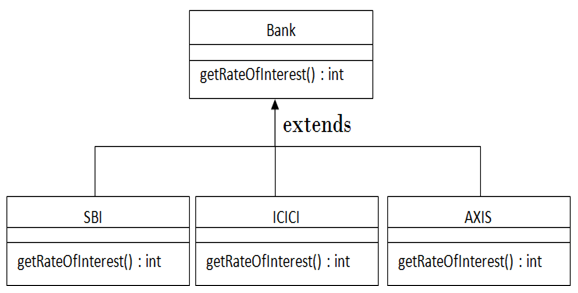
In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method is same and there is IS-A relationship between the classes, so there is method overriding.

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");}
7. public static void main(String args[]){
8. Bike obj = new Bike();
9. obj.run();
10. }

Output:Bike is running safely

**Real example of Java Method Overriding**

Consider a scenario, Bank is a class that provides functionality to get rate of interest. But, 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. }
5. class SBI extends Bank{
6. int getRateOfInterest(){return 8;}
7. }
9. class ICICI extends Bank{
10. int getRateOfInterest(){return 7;}
11. }
12. class AXIS extends Bank{
13. int getRateOfInterest(){return 9;}
14. }
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

**Can we override static method?**

|  |
| --- |
| No, static method cannot be overridden. It can be proved by runtime polymorphism |

**Why we cannot override static method?**

because static method is bound with class whereas instance method is bound with object. **Static** belongs to class area and **instance** belongs to heap area.

**Can we override java main method?**

No, because main is a static method.

**What is the difference between method Overloading and Method Overriding?**

There are three basic differences between the method overloading and method overriding. They are as follows:

|  |  |
| --- | --- |
| **Method Overloading** | **Method Overriding** |
| 1) Method overloading is used to increase the readability of the program. | Method overriding is used to provide the specific implementation of the method that is already provided by its super class. |
| 2) method overloading is performed within a class. | Method overriding occurs in two classes that have IS-A relationship. |
| 3) In case of method overloading parameter must be different. | In case of method overriding parameter must be same. |

In the previous chapter, we talked about super classes and sub classes. If a class inherits a method from its super class, then there is a chance to override the method provided that it is not marked final.

**The benefit of overriding is**: ability to define a behavior that's specific to the subclass type which means a subclass can implement a parent class method based on its requirement.

**In object-oriented terms, overriding means to override the functionality of an existing method.**

## Example:

Let us look at an example.

class Animal{

public void move(){

System.out.println("Animals can move");

}

}

class Dog extends Animal{

public void move(){

System.out.println("Dogs can walk and run");

}

}

public class TestDog{

public static void main(String args[]){

Animal a = new Animal(); // Animal reference and object

Animal b = new Dog(); // Animal reference but Dog object

a.move();// runs the method in Animal class

b.move();//Runs the method in Dog class

}

}

This would produce the following result:

Animals can move

Dogs can walk and run

In the above example, you can see that the even though **b** is a type of Animal it runs the move method in the Dog class. The reason for this is: In compile time, the check is made on the reference type. However, in the runtime, JVM figures out the object type and would run the method that belongs to that particular object.

Therefore, in the above example, the program will compile properly since Animal class has the method move. Then, at the runtime, it runs the method specific for that object.

Consider the following example :

class Animal{

public void move(){

System.out.println("Animals can move");

}

}

class Dog extends Animal{

public void move(){

System.out.println("Dogs can walk and run");

}

public void bark(){

System.out.println("Dogs can bark");

}

}

public class TestDog{

public static void main(String args[]){

Animal a = new Animal(); // Animal reference and object

Animal b = new Dog(); // Animal reference but Dog object

a.move();// runs the method in Animal class

b.move();//Runs the method in Dog class

b.bark();

}

}

This would produce the following result:

TestDog.java:30: cannot find symbol

symbol : method bark()

location: class Animal

b.bark();

^

This program will throw a compile time error since b's reference type Animal doesn't have a method by the name of bark.

## Rules for method overriding:

* The argument list should be exactly the same as that of the overridden method.
* The return type should be the same or a subtype of the return type declared in the original overridden method in the superclass.
* The access level cannot be more restrictive than the overridden method's access level. For example: if the superclass method is declared public then the overridding method in the sub class cannot be either private or protected.
* Instance methods can be overridden only if they are inherited by the subclass.
* A method declared final cannot be overridden.
* A method declared static cannot be overridden but can be re-declared.
* If a method cannot be inherited, then it cannot be overridden.
* A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.
* A subclass in a different package can only override the non-final methods declared public or protected.
* An overriding method can throw any uncheck exceptions, regardless of whether the overridden method throws exceptions or not. However the overriding method should not throw checked exceptions that are new or broader than the ones declared by the overridden method. The overriding method can throw narrower or fewer exceptions than the overridden method.
* Constructors cannot be overridden.