Method Overriding in Java

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 subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

**Usage 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 Java 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.

class Vehicle{

void run(){System.out.println("Vehicle is running");}

}

class Bike extends Vehicle{

public static void main(String args[]){

Bike obj = new Bike();

obj.run();

} }

**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.

class Vehicle{

void run(){System.out.println("Vehicle is running");}

}

class Bike2 extends Vehicle{

void run(){System.out.println("Bike is running safely");}

public static void main(String args[]){

Bike2 obj = new Bike2();

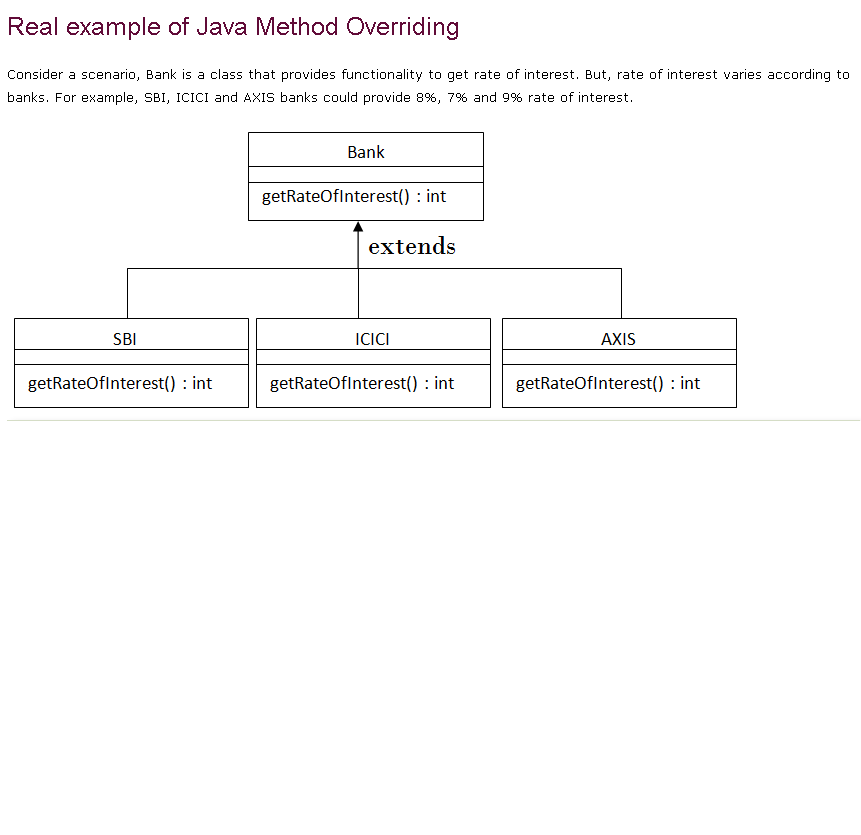
obj.run();

}

**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.



class Bank{

int getRateOfInterest(){return 0;}

}

class SBI extends Bank{

int getRateOfInterest(){return 8;}

}

class ICICI extends Bank{

int getRateOfInterest(){return 7;}

}

class AXIS extends Bank{

int getRateOfInterest(){return 9;}

}

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

**Q) Can we override static method?**

No, static method cannot be overridden, 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.

**Q) Can we override java main method?**

No, because main is a static method.

**Q) Difference between method Overloading and Method Overriding in java**

|  |  |  |
| --- | --- | --- |
| **No.** | **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 class*. | Method overriding occurs *in two classes* that  have IS-A (inheritance) relationship. |
| 3) | In case of method overloading, *parameter must be different*. | In case of method overriding, *parameter must*  *be same*. |
| 4) | Method overloading is the example of *compile time polymorphism*. | Method overriding is the example of *run time*  *polymorphism*. |
| 5) | In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter. | *Return type must be same or covariant* in  method overriding. |

**Java Method Overloading example**

//A class for adding upto 5 numbers

class Sum

{

int add(int n1, int n2)

{

return n1+n2;

}

int add(int n1, int n2, int n3)

{

return n1+n2+n3;

}

int add(int n1, int n2, int n3, int n4)

{

return n1+n2+n3+n4;

}

int add(int n1, int n2, int n3, int n4, int n5)

{

return n1+n2+n3+n4+n5;

}

public static void main(String args[])

{

Sum obj = new Sum();

System.out.println("Sum of two numbers: "+obj.add(20, 21));

System.out.println("Sum of three numbers: "+obj.add(20, 21, 22));

System.out.println("Sum of four numbers: "+obj.add(20, 21, 22, 23));

System.out.println("Sum of five numbers: "+obj.add(20, 21, 22, 23, 24));

}

}

**Output:**

Sum of two numbers: 41

Sum of three numbers: 63

Sum of four numbers: 86

Sum of five numbers: 110

**Java Method Overriding example**

class CarClass

{

public int speedLimit()

{

return 100;

}}

class Ford extends CarClass

{

public int speedLimit()

{

return 150;

}

public static void main(String args[])

{

CarClass obj = new Ford();

int num= obj.speedLimit();

System.out.println("Speed Limit is: "+num);

}

}

**Output:**

Speed Limit is: 150

**Java access modifiers with method overriding**

If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

class A{

protected void msg(){System.out.println("Hello java");}

}

public class Simple extends A{

void msg(){System.out.println("Hello java");}//C.T.Error

public static void main(String args[]){

Simple obj=new Simple();

obj.msg();

} }

**Examples**

**Method Overriding in dynamic method dispatch**

Dynamic method dispatch is a technique which enables us to assign the base class reference to a child class object. As you can see in the below example that the base class reference is assigned to child class object.

class ABC{

public void disp()

{

System.out.println("disp() method of parent class");

}

public void abc()

{

System.out.println("abc() method of parent class");

} }

class Test extends ABC{

public void disp(){

System.out.println("disp() method of Child class");

}

public void xyz(){

System.out.println("xyz() method of Child class");

}

public static void main( String args[]) {

//Parent class reference to child class object

ABC obj = new Test();

obj.disp();

obj.abc();

}}

**Output:**

disp() method of Child class

abc() method of parent class

**Note**: In dynamic method dispatch the object can call the overriding methods of child class and all the non-overridden methods of base class but it cannot call the methods which are newly declared in the child class. In the above example the object obj was able to call the disp()(overriding method) and abc()(non-overridden method of base class). However if you try to call the xyz() method (which has been newly declared in Test class) [obj.xyz()] then it would give compilation error with the following message:

**Exception in thread "main" java.lang.Error: Unresolved compilation**

**problem: The method xyz() is undefined for the type ABC**

**\*\*\*\*\*\*\*\***

**Rules of method overriding in Java**

1) Argument list: The argument list of overriding method must be same as that of the method in parent class. The data types of the arguments and their sequence should be maintained as it is in the overriding method.

2) Access Modifier: The Access Modifier of the overriding method (method of subclass) cannot be more restrictive than the overridden method of parent class. For e.g. if the Access Modifier of base class method is public then the overriding method (child class method ) cannot have private, protected and default Access modifier as all of the three are more restrictive than public.

For e.g. This is not allowed as child class disp method is more restrictive(protected) than base class(public)

class MyBaseClass{

public void disp()

{

System.out.println("Parent class method");

}

}

class MyChildClass extends MyBaseClass{

protected void disp(){

System.out.println("Child class method");

}

public static void main( String args[]) {

MyChildClass obj = new MyChildClass();

obj.disp();

}}

Output:

Exception in thread "main" java.lang.Error: Unresolved compilation

problem: Cannot reduce the visibility of the inherited method from MyBaseClass

However this is perfectly valid scenario as public is less restrictive than protected. Same access modifier is also a valid one.

class MyBaseClass{

protected void disp()

{

System.out.println("Parent class method");

}

}

class MyChildClass extends MyBaseClass{

public void disp(){

System.out.println("Child class method");

}

public static void main( String args[]) {

MyChildClass obj = new MyChildClass();

obj.disp();

}

}

Output:

Child class method

3) private, static and final methods cannot be overridden as they are local to the class. However static methods can be re-declared in the sub class, in this case the sub-class method would act differently and will have nothing to do with the same static method of parent class.

4) Overriding method (method of child class) can throw any unchecked exceptions, regardless of whether the overridden method(method of parent class) throws any exception or not. However the overriding method should not throw checked exceptions that are new or broader than the ones declared by the overridden method. We will discuss this in detail with example in the upcoming tutorial.

5) Binding of overridden methods happen at runtime which is known as dynamic binding.

6) If a class is extending an abstract class or implementing an interface then it has to override all the abstract methods unless the class itself is a abstract class.

**Super keyword in Overriding**

super keyword is used for calling the parent class method/constructor. super.methodname() calling the specified method of base class while super() calls the constructor of base class. Let’s see the use of super in Overriding.

class ABC{

public void mymethod()

{

System.out.println("Class ABC: mymethod()");

}

}

class Test extends ABC{

public void mymethod(){

//This will call the mymethod() of parent class

super.mymethod();

System.out.println("Class Test: mymethod()");

}

public static void main( String args[]) {

Test obj = new Test();

obj.mymethod();

}

}

Output:

Class ABC: mymethod()

Class Test: mymethod()

