

Method Overriding in Java:

1. If the child class implements the same method present in the parent class again, it is known as method overriding.

2. Method overriding helps us to classify a behavior that is specific to the child class.

3. The subclass can override the method of the parent class only when the method is not declared as final.

4. Example :

- In the below code, we've created two classes: class A & class B.

- Class B is inheriting class A.

- In the main() method, we've created one object for both classes. We're needing the method() method on class A and B objects

- separately, but the output is the same because the method() is defined in the parent class, i.e., class A.

```
1 class A {
2     public void method1(){
3         System.out.println("I am method 1 of class A");
4     }
5 }
6
7 class B extends A {
8 }
9
10 public class CodeXam {
11     public static void main(String[] args) {
12         A a = new A();
13         a.method1();
14
15         B b = new B();
16         b.method1();
17     }
18 }
```

Run CodeXam (1) ×

I am method 1 of class A
I am method 1 of class A

Now, let's see how we can override the method() for class B :

```
1 class A {
2     public void method1(){
3         System.out.println("I am method 1 of class A");
4     }
5 }
6
7 class B extends A {
8     @Override
9     public void method1(){
10        System.out.println("I am method 1 of class B");
11    }
12 }
13
14 public class CodeXam {
15     public static void main(String[] args) {
16         A a = new A();
17         a.method1();
18
19         B b = new B();
20         b.method1();
21     }
22 }
```

Run CodeXam ×

I am method 1 of class A
I am method 1 of class B

It's not mandatory to write @Override

However, you should write override to avoid confusion between the two methods. It won't be an override if you pass parameters to any method, so you should write override instead.

Overriding vs Overloading in Java

```
1 package com.journaldev.examples;
2 import java.util.Arrays;
3
4 public class Processor {
5
6     public void process(int i, int j) {
7         System.out.printf("Processing two integers:%d, %d", i, j);
8     }
9
10    public void process(int[] ints) {
11        System.out.println("Adding integer arrays" + Arrays.toString(ints));
12    }
13
14    public void process(Object[] objs) {
15        System.out.println("Adding integer arrays" + Arrays.toString(objs));
16    }
17 }
18
19 class MathProcessor extends Processor {
20
21     @Override
22     public void process(int i, int j) {
23         System.out.println("Sum of integers is " + (i + j));
24     }
25
26     @Override
27     public void process(int[] ints) {
28         int sum = 0;
29         for (int i : ints) {
30             sum += i;
31         }
32         System.out.println("Sum of integer array elements is " + sum);
33     }
34 }
35
36 }
```

Overloading: Same Method Name but different parameters in the same class

Overriding: Same Method Signature in both superclass and child class

1. What is Overloading and Overriding?

When two or more methods in the same class have the same name but different parameters, it's called Overloading. When the method signature (name and parameters) are the same in the superclass and the child class, it's called Overriding.

2. Overriding vs Overloading

Overriding implements Runtime Polymorphism whereas Overloading implements Compile time polymorphism.

The method Overriding occurs between superclass and subclass. Overloading occurs between the methods in the same class.

Overriding methods have the same signature i.e. same name and method arguments. Overloaded method names are the same but the parameters are different.

With Overloading, the method to call is determined at the compile-time. With overriding, the method call is determined at the runtime based on the object type.

If overriding breaks, it can cause serious issues in our program because the effect will be visible at runtime. Whereas if overloading breaks, the compile-time error will come and it's easy to fix.

3. Overloading and Overriding Example

Here is an example of overloading and overriding in a Java program.

```
1 import java.util.Arrays;
2
3 class Processor {
4     public void process(int i, int j) {
5         System.out.printf("Processing two integers:%d, %d", i, j);
6     }
7
8     public void process(int[] ints) {
9         System.out.println("Adding integer array:" + Arrays.toString(ints));
10    }
11
12    public void process(Object[] objs) {
13        System.out.println("Adding integer array:" + Arrays.toString(objs));
14    }
15 }
16
17 class MathProcessor extends Processor {
18
19     @Override
20     public void process(int i, int j) {
21         System.out.println("Sum of integers is " + (i + j));
22     }
23
24     @Override
25     public void process(int[] ints) {
26         int sum = 0;
27         for (int i : ints) {
28             sum += i;
29         }
30         System.out.println("Sum of integer array elements is " + sum);
31     }
32 }
33
34 public class CodeXam {
35
36     public static void main(String[] args) {
37         Processor processor = new MathProcessor();
38         processor.process(1, 2);
39         processor.process(new int[]{1, 2, 3, 4, 5});
40         processor.process(new Object[]{1, 2, 3, 4, 5});
41     }
42 }
```

Run CodeXam ×

Sum of integers is 3
Sum of integer array elements is 15
Adding integer array:[1, 2, 3, 4, 5]