Java Method Overloading

In this article, you'll learn about method overloading and how you can achieve it in Java with the help of examples.

In Java, two or more methods may have the same name if they differ in parameters (different number of parameters, different types of parameters, or both). These methods are called overloaded methods and this feature is called method overloading. For example:

void func() { ... } void func(int a) { ... } float func(double a) { ... } float func(int a, float b) { ... }

Here, the `func()` method is overloaded. These methods have the same name but accept different arguments.

Note: The return types of the above methods are not the same. It is because method overloading is not associated with return types. Overloaded methods may have the same or different return types, but they must differ in parameters.

Why method overloading?

Suppose, you have to perform the addition of given numbers but there can be any number of arguments (let's say either 2 or 3 arguments for simplicity).

In order to accomplish the task, you can create two methods `sum2num(int, int)` and `sum3num(int, int, int)` for two and three parameters respectively. However, other programmers, as well as you in the future may get confused as the behavior of both methods are the same but they differ by name.

The better way to accomplish this task is by overloading methods. And, depending upon the argument passed, one of the overloaded methods is called. This helps to increase the readability of the program.

How to perform method overloading in Java?

Here are different ways to perform method overloading:

1. Overloading by changing the number of parameters

```
class MethodOverloading {
    private static void display(int a){
        System.out.println("Arguments: " + a);
    }

    private static void display(int a, int b){
        System.out.println("Arguments: " + a + " and " + b);
    }

    public static void main(String[] args) {
        display(1);
        display(1, 4);
    }
}
```

#div-gpt-ad-Programizcom37046 {display:none; width: 728px; height: 90px; } #div-gpt-ad-Programizcom36796 {display: block;} @media(min-width: 992px) { #div-gpt-ad-Programizcom37046 {display: block;} #div-gpt-ad-Programizcom36796 {display: none;}}

Output:

Arguments: 1 Arguments: 1 and 4

2. Method Overloading by changing the data type of parameters

```
class MethodOverloading {

    // this method accepts int
    private static void display(int a){
        System.out.println("Got Integer data.");
    }

    // this method  accepts String object
    private static void display(String a){
        System.out.println("Got String object.");
    }

    public static void main(String[] args) {
        display(1);
        display("Hello");
    }
}
```

Output:

Got Integer data. Got String object.

Here, both overloaded methods accept one argument. However, one accepts the argument of type `int` whereas other accepts `String` object.

Let’s look at a real-world example:

```
class HelperService {

    private String formatNumber(int value) {
        return String.format("%d", value);
    }

    private String formatNumber(double value) {
        return String.format("%.3f", value);
    }

    private String formatNumber(String value) {
        return String.format("%.2f", Double.parseDouble(value));
    }

    public static void main(String[] args) {
        HelperService hs = new HelperService();
        System.out.println(hs.formatNumber(500));
        System.out.println(hs.formatNumber(89.9934));
        System.out.println(hs.formatNumber("550"));
    }
}
```

When you run the program, the output will be:

500 89.993 550.00

Note: In Java, you can also overload constructors in a similar way like methods.

Recommended Reading: Java Constructor Overloading

Important Points

- Two or more methods can have the same name inside the same class if they accept different arguments. This feature is known as method overloading.
- Method overloading is achieved by either:
  - changing the number of arguments.
  - or changing the data type of arguments.
- It is not method overloading if we only change the return type of methods. There must be differences in the number of parameters.