Method Overloading

In Java, two or more methods can have 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, func() method is overloaded. These methods have same name but accept different arguments.
Notice that, the return type of these methods is not same. Overloaded methods may or may not have different return type, but they must differ in parameters they accept.

Why method overloading?
Suppose, you have to perform addition of the 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 future may get confused as the behavior of both methods is 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 readability of the program.

How to perform method overloading in Java?
Here are different ways to perform method overloading:
1. Overloading by changing number of arguments

```
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);
    }
}
```

When you run the program, the output will be:

```
Arguments: 1
Arguments: 1 and 4
```

2. By changing the datatype 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");
}
```

When you run the program, the output will be:

```
Got Integer data.
Got String object.
```

Here, both overloaded methods accept one argument. However, one accepts 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
```

Important Points

- Two or more methods can have 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 datatype of arguments.
- Method overloading is not possible by changing the return type of methods.