Method Overloading in Java

Overloading in Java

In this tutorial, we shall learn about Overloading in Java. Overloading is the ability to use same name for different methods with different set of parameters. We shall go through some Java Example Programs in detail to understand overloading in Java.

Overloading is a way to realize Polymorphism in Java.

If a method can expect different types of inputs for the same functionality it implements, implementing different methods (with different method names) for each scenario may complicate the readability of code.

For example, we need a functionality to add two numbers. Now these two numbers could be integer values, float values, long values or double values. If we implement methods like addTwoInts(int a, int b), addTwoFloats(float a, float b), addIntFloat(int a, float b), etc., it becomes very difficult to use these methods in other classes because of lack of readability. Java provides the ability to overload a method based on the type of arguments passed in the function call, provided the methods have a same name.

How to implement Overloading in Java?

Following are the rules to implement Overloading of methods.

1. All the methods that take park in Overloading should have the same name.
2. No two methods should have the same set of parameters. They should differ either in
    - the number of parameters they have in their definition or
    - the type of parameters they have in their definition

If a method is not found for a set of arguments, the compiler does **Type Promotion** and checks for a suitable method definition. We shall learn in detail about Type Promotion with an example in due course of this tutorial.

Note : Also, please note that, different return types do not make two methods different with respect to Overloading.

Example 1 – Overloading in Java

We shall use the same example of addition to demonstrate Overloading in Java.

Calculation.java

```
/**
 * Example program to demonstrate Overloading in Java
 */
public class Calculation {

    public static void main(String[] args) {
        Calculation calc = new Calculation();
        System.out.println(calc.add(4, 8)); // add(int a, int b) is loaded
        System.out.println(calc.add(4, 8.5));   // add(int a, float b)  is loaded
        System.out.println(calc.add("25", "85"));   // add(String a, String b) is loaded
        System.out.println(calc.add(25, 8888888888888888888888.0));    // add(int a, do
        System.out.println(calc.add(25));   // add(int a) is loaded
    }

    public int add(int a, int b){
        System.out.println("add(int a, int b) is called");
        return a+b;
    }

    public int add(String a, String b){
        System.out.println("add(String a, String b) is called");
        return Integer.parseInt(a)+Integer.parseInt(b);
    }

    public float add(int a, float b){
        System.out.println("add(int a, float b) is called");
        return a+b;
    }

    public double add(int a, double b){
        System.out.println("add(int a, double b) is called");
        return a+b;
    }

    public int add(int a){
        System.out.println("add(int a) is called");
        return a+1;
    }
}
```

Run the above program, and you shall get the following output in console.

Output

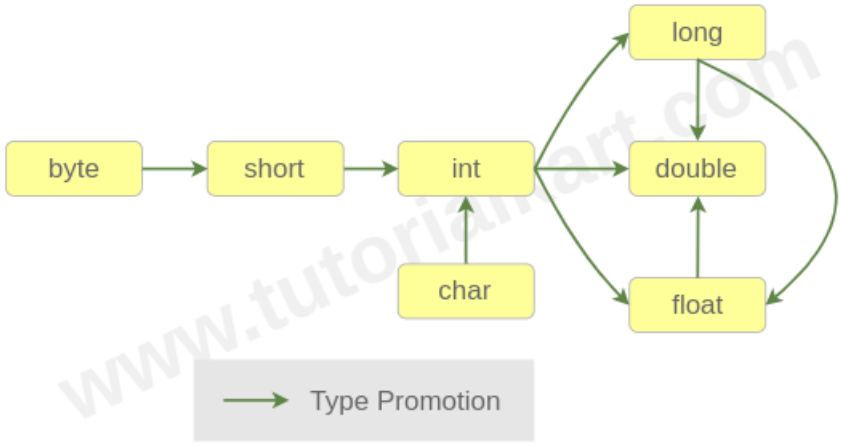
```
add(int a, int b) is called
12
add(int a, double b) is called
12.5
add(String a, String b) is called
110
add(int a, double b) is called
8.888888888888889E22
add(int a) is called
26
```

Now we shall cross verify the rules that we already stated with the above program.

1. All the five methods that take part in overloading have same method name “add”.
2. No two definitions of add function have same set of arguments. One add method have (int, int), another has (int, float), etc.

Type promotion during method Overloading

If a method is not found for a set of arguments, the compiler does **Type Promotion** and checks for a suitable method definition. Following diagram shows different type promotions that are possible.



If there is forward path between two data types, then the starting datatype in the path can be type promoted to the datatype at end of the path.

- byte can be type promoted to short.
- short can be type promoted to int.
- **Implication can be applied in type promotion**. i.e., if byte can be promoted to short and short can be promoted to int, then byte can be promoted to int. Also byte can be promoted to long, double or float.

Example 2 – Type Promotion – Overloading

We shall use the same Calculation class, but remove the definition add(int, int). As we have add(int, float), even we make a call add(int, int), as int can be promoted to float, double or long, we have add(int, float) and add(int, double) which are potential overloadable methods.

Calculation.java

```
/**
 * Example program to demonstrate Overloading in Java with Type Promotion
 */
public class Calculation {

    public static void main(String[] args) {
```

```
    public static void main(String[] args) {
        Calculation calc = new Calculation();
        System.out.println(calc.add(4, 8));
        System.out.println(calc.add(4, 8.5));
        System.out.println(calc.add("25", "85"));
        System.out.println(calc.add(25, 8888888888888888888888.0));
        System.out.println(calc.add(25));
    }

    public int add(String a, String b){
        System.out.println("add(String a, String b) is called");
        return Integer.parseInt(a)+Integer.parseInt(b);
    }

    public float add(int a, float b){
        System.out.println("add(int a, float b) is called");
        return a+b;
    }

    public double add(int a, double b){
        System.out.println("add(int a, double b) is called");
        return a+b;
    }

    public int add(int a){
        System.out.println("add(int a) is called");
        return a+1;
    }
}
```

Run the above program and you shall get the following output in console.

Output

```
add(int a, float b) is called
12.0
add(int a, double b) is called
12.5
add(String a, String b) is called
110
add(int a, double b) is called
8.888888888888889E22
add(int a) is called
26
```

Int has been promoted to Float.

Now, we shall remove add(int, float) and see the result.

Calculation.java

```
/**
 * Example program to demonstrate Overloading in Java
 */
public class Calculation {
```

```
    public static void main(String[] args) {
        Calculation calc = new Calculation();
        System.out.println(calc.add(4, 8));
        System.out.println(calc.add(4, 8.5));
        System.out.println(calc.add("25", "85"));
        System.out.println(calc.add(25, 8888888888888888888888.0));
        System.out.println(calc.add(25));
    }

    public int add(String a, String b){
        System.out.println("add(String a, String b) is called");
        return Integer.parseInt(a)+Integer.parseInt(b);
    }

    public double add(int a, double b){
        System.out.println("add(int a, double b) is called");
        return a+b;
    }

    public int add(int a){
        System.out.println("add(int a) is called");
        return a+1;
    }
}
```

Run the above program.

```
add(int a, double b) is called
12.0
add(int a, double b) is called
12.5
add(String a, String b) is called
110
add(int a, double b) is called
8.888888888888889E22
add(int a) is called
26
```

For both add(int, int) and add(int, float); **add(int, double)** has been called i.e., int and float are promoted to double.

You may experiment with other type promotions and verify the results with the type promotions presented in the picture.

Conclusion

In this Java Tutorial, we learned about Overloading in Java with an Example program of adding two numbers for different set of input parameters. Also, we learned type promotions that are possible with different data types in Java for overloading methods.

In our next tutorial, we shall learn about Overriding in Java.

Java Tutorial