Overloading Methods

by Sophia

WHAT'S COVERED

In this lesson, you will learn about overriding methods when using loops. Specifically, this lesson covers:

Table of Contents

1. Overloading Methods

It is useful to have different versions of a method that have the same name but take different numbers and types of parameters, when designing methods in Java. When there is more than one version of a method with the same name that take different parameters, including different numbers or types of parameters, the method is said to be **overloaded**. It is important to note that only the method's signature, or the method name and parameters, play a role in method overloading. The return type is not considered.

CONCEPT TO KNOW

It is not possible to have two versions of a method with the same name that differ only in the return type.

We have previously encountered this simple method that takes a single String parameter:

EXAMPLE

```
static String sayHello(String name) {
  return "Hello, " + name;
}
```

We have also seen a different version of the sayHello() method that takes two parameters, a String for the name and an int for the number of repetitions:

EXAMPLE

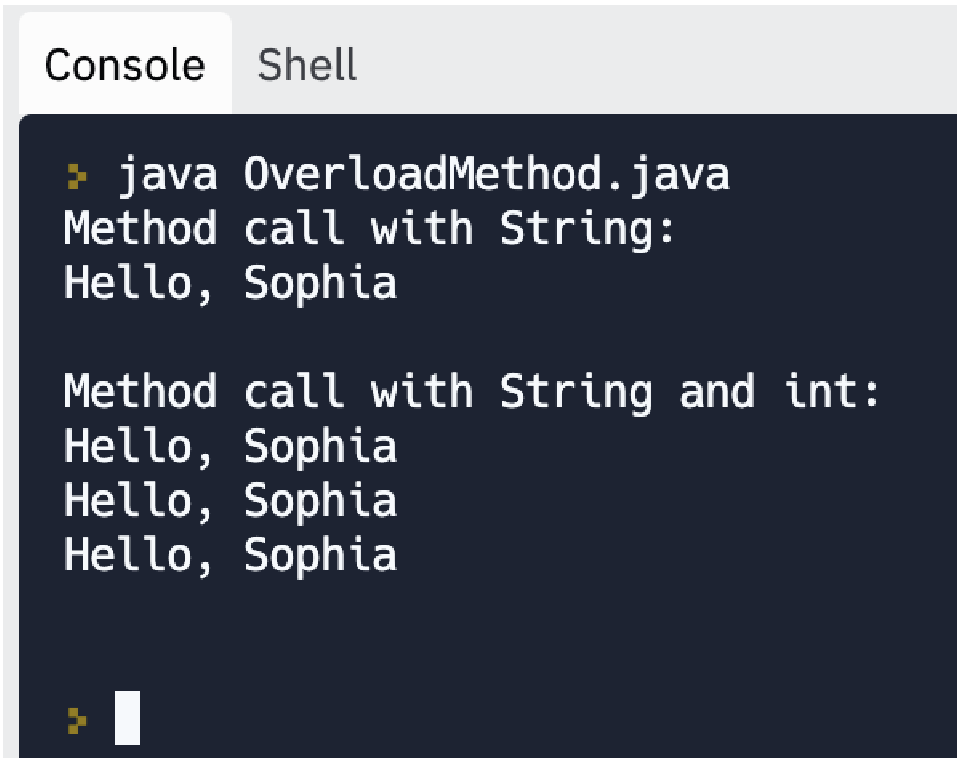
```
static String sayHello(String name, int count) {
    // Local variable to assemble greeting
    String greeting = "";
    for(int i = 0; i < count; i++) {
      greeting += "Hello, " + name + "\n";
    }
    return greeting;
  }
```

When working with these methods in lesson 2.3.1, you found that they were in separate programs. Since they have different method signatures, or different numbers of parameters, they can be included in the same program.

The Java compiler determines which version of the method intended to be called based on the number and types of the arguments passed:

```
class OverloadMethod {
  public static void main(String[] args) {
    System.out.println("Method call with String: ");
    String result = sayHello("Sophia");
    System.out.println(result + "\n");

    System.out.println("Method call with String and int: ");
    result = sayHello("Sophia", 3);
    System.out.println(result + "\n");
  }

  // Method with 1 String parameter

}
```

If you type in the code above (in a file named OverloadMethod.java) and run it in Replit, the results should look like this:



```
> java OverloadMethod.java
Method call with String:
Hello, Sophia

Method call with String and int:
Hello, Sophia
Hello, Sophia
Hello, Sophia


>
```

The compiler looks for a version of the sayHello() method that has the parameters that match the arguments passed when the method is called.

Notice the call on this line:

↪ EXAMPLE

```
String result = sayHello("Sophia");
```

It invokes the version of sayHello() with this signature:

↪ EXAMPLE

```
sayHello(String name)
```

Notice the call to sayHello() on this line:

↪ EXAMPLE

```
result = sayHello("Sophia", 3);
```

It matches the overload of the sayHello() method with this signature:

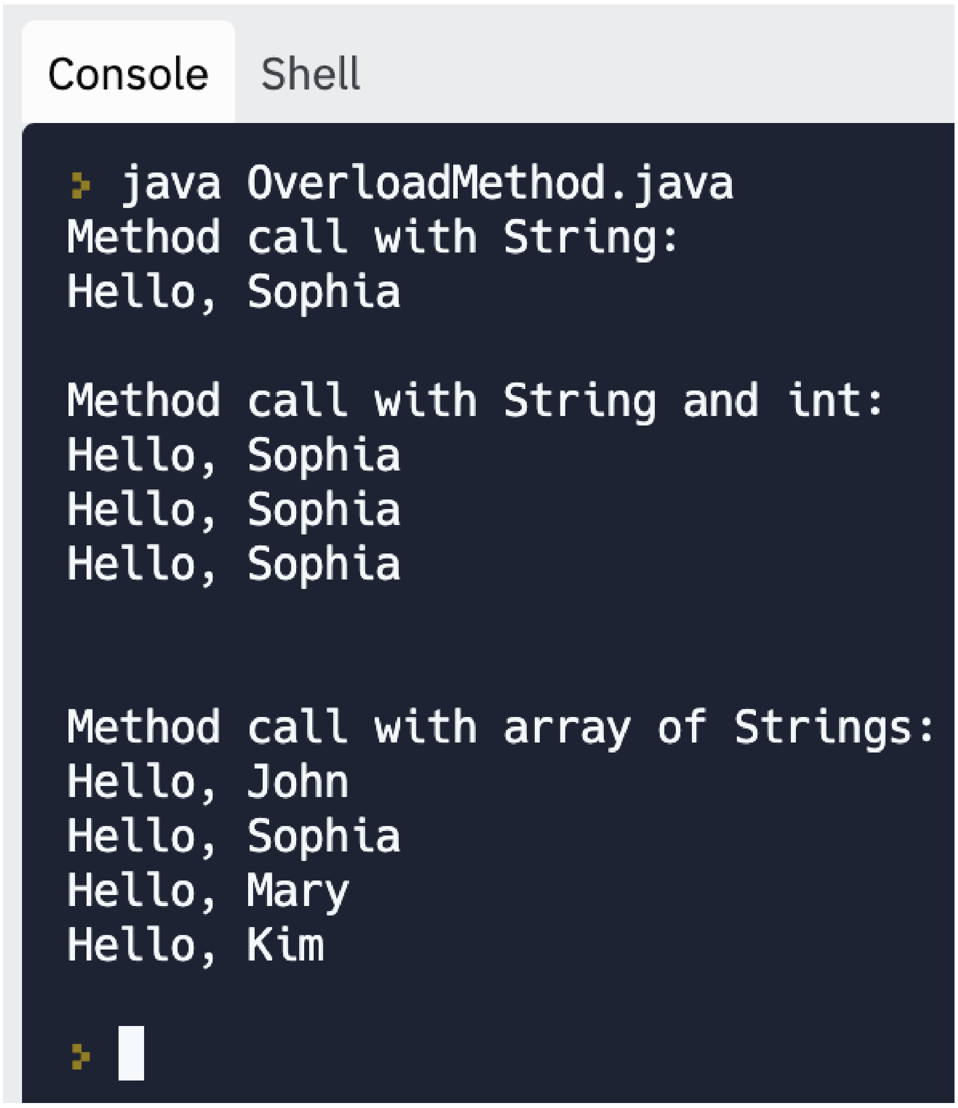
↪ EXAMPLE

```
sayHello(String name, int count)
```

These two overloads of the sayHello() method differ in the number of parameters. An overload can be written of the method that takes an array of String values, rather than a plain String. While this version has only one parameter, the parameter is a String[] rather than a single String. The parameter is of a different type, which is an array of Strings rather than a String. We can call this overload of the method by passing an array of Strings as the argument.

This version of the program demonstrates the use of these three overloads in a single program:

```
class OverloadMethod {
  public static void main(String[] args) {
    System.out.println("Method call with String: ");
    String result = sayHello("Sophia");
    System.out.println(result + "\n");

    System.out.println("Method call with String and int: ");
    result = sayHello("Sophia", 3);
    System.out.println(result + "\n");

    System.out.println("Method call with array of Strings: ");
    String[] firstNames = {"John", "Sophia", "Mary", "Kim"};
    result = sayHello(firstNames);
    System.out.println(result);
  }

  // Method with 1 String parameter
  static String sayHello(String name) {
    return "Hello, " + name;
  }

  // Method with 2 parameters (String & int)
  static String sayHello(String name, int count) {
    // Local variable to assemble greeting
    String greeting = "";
    for(int i = 0; i < count; i++) {
      greeting += "Hello, " + name + "\n";
    }
    return greeting;
  }

  // Method with 1 array of Strings parameter
  static String sayHello(String[] names) {
    String greeting = "";
    for(String name : names) {
      greeting += "Hello, " + name + "\n";
    }
    return greeting;
  }
}
```
This program with three overloads of the `sayHello()` method produces these results (assuming the code is saved in a file named OverloadMethod.java:



```
> java OverloadMethod.java
Method call with String:
Hello, Sophia

Method call with String and int:
Hello, Sophia
Hello, Sophia
Hello, Sophia


Method call with array of Strings:
Hello, John
Hello, Sophia
Hello, Mary
Hello, Kim

>
```

CONCEPT TO KNOW

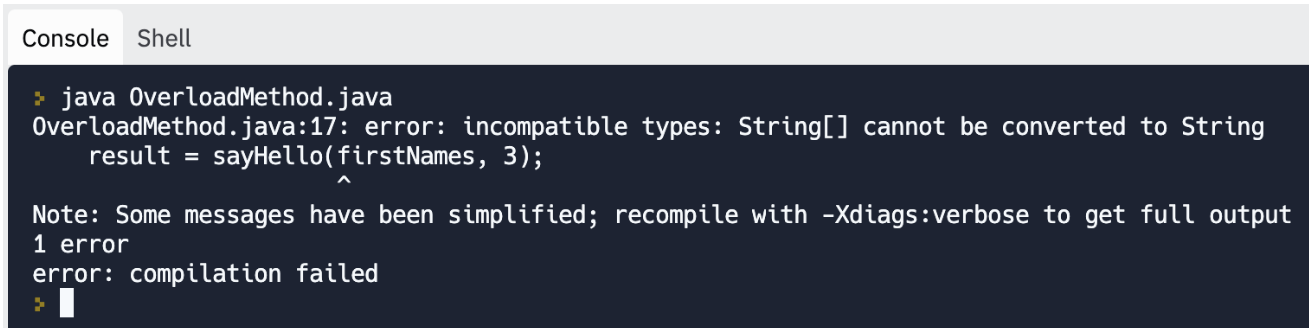
The programmer is responsible for providing the overloads of the method. If the program tries to call a method with parameters that don't correspond to one of the overloads, the program ends with an exception error.

Consider if the code above included a call that passes an array and an int using a call like this:

EXAMPLE

```
result = sayHello(firstNames, 3);
```

The result is this:



```
> java OverloadMethod.java
OverloadMethod.java:17: error: incompatible types: String[] cannot be converted to String
    result = sayHello(firstNames, 3);
                      ^
Note: Some messages have been simplified; recompile with -Xdiags:verbose to get full output
1 error
error: compilation failed
>
```

While it may be difficult to understand what this error is trying to communicate, the first line of the output indicates that the compiler can't convert an array of Strings to a plain String. The compiler "sees" that there is an overload of the method with two parameters, but it can't convert an array of Strings to a String. There is no appropriate match.

The method definitions below were based on the code above:

EXAMPLE

```
static String sayHello(String name)
static String sayHello(String name, int count)
static String sayHello(String[] names)
```

TERM TO KNOW

Overloaded
A method is overloaded if different versions of the method exist that differ in the number or type of parameters in the method's signature.

SUMMARY

In this lesson, you learned about **overloading methods**. You also learned that there are different versions of the same method that can be called based on the signature of the method. This means that the overloads can have different types and numbers of parameters. You learned that when the return type is not part of a method's signature, it is not possible to have overloads that differ only by their return types. Finally, you learned that methods that need to return different data types need to have different names.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source py4e.com/html3/

TERMS TO KNOW

Overloaded
A method is overloaded if different versions of the method exist that differ in the number or type of

parameters in the method's signature.