

The Return Statement

by Sophia



WHAT'S COVERED

In this lesson, you will learn about the return statement in methods. Specifically, this lesson covers:

Table of Contents

- [1. Return Statements](#)
- [2. Return void](#)
- [3. Return a Value](#)

1. Return Statements

When a method is done doing its work, it needs to return control of the program flow to the portion of the called, that called it. To understand how methods return the program flow to the caller, you will need to discern methods that do their work and don't need to send back a value. Next, you'll need to consider methods that are designed to do their work in order to return the results of their work. The Java data type that specifies the kind of value returned by a method is called the method's **return type**.

It is important to keep in mind that a method may return no value, or only one value or object. When the method does not return a value, the return type is said to be void. In cases where the method returns a value, the return type is the data type of the value returned.

➞ EXAMPLE

int, double, char, String or data type.



TERMS TO KNOW

return

The reserved keyword return is used to exit a method and return a value, if the method returns a value.

Return Type

The data type of the value returned by a method.

2. Return void

Some methods print out a greeting without needing to return anything.

A Simple Method:

➞ EXAMPLE

```
// A simple method that doesn't need to return anything
public static void printGreeting(String name) {
    System.out.println("Hello, " + name);
    return;
}
```

This method calls `System.out.println()` itself, so there is no further work to be done when the method is finished. The keyword `void` to the left of the method's name indicates that the method does not return a value.

The return statement without a value indicates that the method returns to the caller after printing out the greeting, but it also makes clear that no value is returned. In a method that returns nothing, it has the return type `void`.

When a method has the return type `void`, there is no need to provide an explicit return statement. In such cases, the control flow in the program returns to the caller when the code in the method is done executing.



TERM TO KNOW

void

The return type `void` indicates that a method does not return a value.

3. Return a Value

Many times, a method will need to return a value to the caller, with the result of running the code in the method. For instance, the simplest version of the `sayHello()` method from the last tutorial was defined like this because it returned a `String` with the greeting assembled by the method.



CONCEPT TO KNOW

The return type is the data type to the left of the name of the method. Keep in mind that `static` is an access modifier that we will discuss later in the course, so look past it for now.

A static `String`:

➞ EXAMPLE

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

```
}
```

This method returns a String with the greeting concatenated with the desired name appended to it.

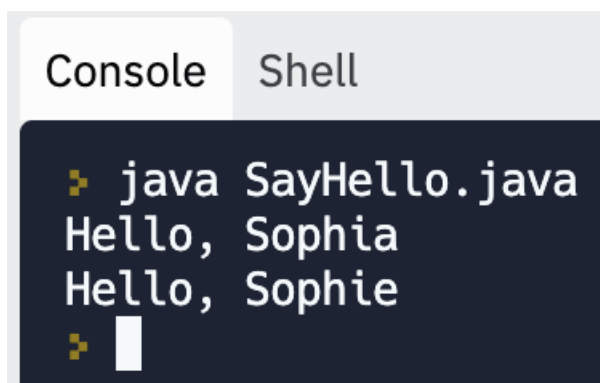


TRY IT

Directions: When calling a method with a returned value, the caller provides a variable to hold the result that is returned or passes the value returned to another method, placing one method call inside of another:

```
class SayHello {  
    public static void main(String[] args) {  
        // The variable to the left of the equal sign holds  
        // the value returned from the method (a String)  
        String greeting = sayHello("Sophia");  
        System.out.println(greeting);  
        // In this call, the String returned from the method  
        // is passed directly to println() for display  
        System.out.println(sayHello("Sophie"));  
    }  
  
    static String sayHello(String name) {  
        return "Hello, " + name;  
    }  
}
```

The output for this should look like this:



```
Console Shell  
java SayHello.java  
Hello, Sophia  
Hello, Sophie  
[ ]
```



REFLECT

This small method returns the String with the greeting text rather than printing it out directly. Why might a programmer prefer a method that returns a value for display rather than printing it out directly in the method itself?

The first line of output is the result of running these two statements on two separate lines of code:

➞ EXAMPLE

```
String greeting = sayHello("Sophia");  
System.out.println(greeting);
```

The second line of output is the result of running this one line of code:

➞ EXAMPLE

```
System.out.println(sayHello("Sophie"));
```

Note that it is also possible to make a call like this where there is no variable to capture the value returned.

The method call is not embedded in another method call:

➞ EXAMPLE

```
sayHello("Sofia");
```



There is little point in doing so, since the `String` returned by `sayHello()` is lost.

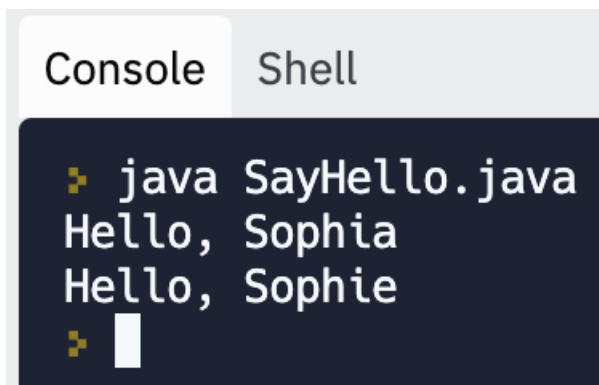


Directions: Add the statement to the code. When you run it, you should notice that it has no visible effect:

```
class SayHello {
    public static void main(String[] args) {
        // The variable to the left of the equal sign holds
        // the value returned from the method (a String)
        String greeting = sayHello("Sophia");
        System.out.println(greeting);
        // In this call, the String returned from the method
        // is passed directly to println() for display
        System.out.println(sayHello("Sophie"));
        // Since the following does not save the returned String
        // to a String variable and since the call is not embedded in
        // another method call, the String is lost
        sayHello("Sofia");
    }

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

This output shows the same two lines as before, with no sign of the third call to `sayHello()`:



```
Console Shell
java SayHello.java
Hello, Sophia
Hello, Sophie

```

REFLECT

When calling a method that returns a value, the code should normally assign the return value to an appropriate variable. Why would overlooking the returned value be unwise?

In addition to returning a single, primitive value, a method can also return multiple values using an array or a collection. Keep in mind that all of the values in the array or collection have to be of the same data type.

Here is an example of a program that includes a method that returns an array of random integers:

```
import java.util.Random;
import java.util.Arrays;

class RandomMethod {
    // Method returns an array of random integers
    // Parameters are maximum of the range (minimum is 1)
    // count is number of random numbers to return
    static int[] getRandomIntegers(int max, int count) {
        // Create random number generator - see the guessing game
        Random randomGenerator = new Random();
        int[] randomNumbers = new int[count];
        for(int i = 0; i < count; i++) {
            // Get a random number in the range from 1 to max
            randomNumbers[i] = randomGenerator.nextInt(max) + 1;
        }
        // Return array to caller
        return randomNumbers;
    }

    public static void main(String[] args) {
        // Get an array of 3 integers in the range from 1 to 100
        int[] results = getRandomIntegers(100, 3);
        System.out.println("Results: " + Arrays.toString(results));
    }
}
```

Here is the output from a sample run of the program:

A terminal window with tabs for 'Console' and 'Shell'. The 'Console' tab is active, showing a command prompt where `java RandomMethod.java` has been executed. The output is `Results: [2, 46, 52]`. A cursor is visible on the line following the output.

It is possible to create a similar method that returns a collection (an `ArrayList`) instead of an array.



Directions: Type in and run this version of the code:

```
import java.util.Random;
import java.util.ArrayList;

class RandomCollectionMethod {
    static ArrayList<Integer> getRandomIntegers(int max, int count) {
        // Create random number generator - see the guessing game
        Random randomGenerator = new Random();
        ArrayList<Integer> randomNumbers = new ArrayList<>();
        for(int i = 0; i < count; i++) {
            // Get a random number in the range from 1 to max
            randomNumbers.add(randomGenerator.nextInt(max) + 1);
        }
        // Return array to caller
        return randomNumbers;
    }

    public static void main(String[] args) {
        // Get an array of 3 integers in the range from 1 to 100
        ArrayList<Integer> results = getRandomIntegers(100, 3);
        System.out.println("Results: " + results.toString());
    }
}
```

A sample run of this version produces similar results (but keep in mind that the numbers are random, so the output will vary):

A terminal window with tabs for 'Console' and 'Shell'. The 'Console' tab is active, showing a command prompt where `java RandomCollectionMethod.java` has been executed. The output is `Results: [71, 95, 32]`. A cursor is visible on the line following the output.



REFLECT

Returning an array or collection of values is a good way to return more than one value from a method, but array values must all be of the same data type. What effect does this fact have on the usefulness of returning an array from a method?



TRY IT

Directions: Go ahead and add the example above into Replit and see if you get the same type of data in the output.



BRAINSTORM

In a method like the one above, do you see advantages to using an array or a collection?



SUMMARY

This tutorial has covered the process of returning methods. Some methods do their work without **returning a value**. You learned that some methods have the **return void** with a type **void**. You also learned that other methods send back the result of the method's work using a **return statement**. Finally, you learned that a return statement can only return one item, but an array or a collection can be used to return multiple values of the same data type.

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

Return Type

The data type of the value returned by a method.

return

The reserved keyword return is used to exit a method and return a value, if the method returns a value.

void

The return type void indicates that a method does not return a value.