Method Parameters and Arguments

by Sophia

WHAT'S COVERED

In this lesson, you will learn about method parameters and arguments that are used when determining needed values. In tutorial 1.3.1, you briefly learned about method parameters and arguments. In this tutorial, you will look further and work with method parameters and arguments in greater detail. Specifically, this lesson covers:

1. Method Parameters

In the introduction to methods provided in tutorial 1.3.1, you saw examples of how methods in Java can include a special kind of variable called a **parameter**. You learned that parameters are used to pass data when calling a method. It is possible to have methods that do not take any parameters. This is because they do not require data from outside of the method to do their work. However, it is more common to create and use methods that take one or more parameters. Parameters for a method are placed in the parentheses that follow the name of the method. The earlier discussion included a simple Java method:

EXAMPLE

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

The method in this case has the name `sayHello`. In the parentheses following the method name, you can see that there is one parameter for passing data to the method. `String` indicates that the parameter must be a Java `String`. The data type is followed by the parameter name, which in this case is `name`. The name of the method and the list of parameters make up the method's signature.

The parameters then serve as variables in the body of the method. This means that the parameters are **variables with local scope** (more commonly called **local variables**). Local variables are accessible from where they are declared to the end of the block. Parameters are "in scope" or accessible between the method signature and the end of the method.

As the code above shows, the name of the parameter can then be used as a variable in the body of the method. The value in the variable name is concatenated at the end of the greeting text.

The method can be called from the `main()` method. A complete sample program looks like this:

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

  public static void main(String[] args) {
    String userName = "Sophia";
    // Pass variable userName as parameter to sayHello()
    String greeting = sayHello(userName);
    // Pass variable greeting to as parameter println()
    System.out.println(greeting);
  }
}
```

The output from running this code looks like this:

```
~/IntrotoJava$ java src/main/java/SayHello.java
Hello, Sophia
~/IntrotoJava$
```

This method's signature includes only one parameter, but the method can be defined to have more. You can add an `int` parameter to indicate the number of repetitions, to repeat the greeting.

Adding an `int` parameter to indicate repetition:

```
public class SayHelloRepeated {
  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;
  }
  public static void main(String[] args) {
    String userName = "Sophia";
    // Pass variable userName as parameter to sayHello()
    String greetingOutput = sayHello(userName, 3);
    // Pass variable greetingOutput to as parameter println()
    System.out.println(greetingOutput);
  }
}
```

The output from this version of the program should look like this:

```
~/IntrotoJava$ java src/main/java/SayHelloRepeated.java
Hello, Sophia
Hello, Sophia
Hello, Sophia

~/IntrotoJava$
```

Rather than printing the same name multiple times, you can use an array as a parameter to pass multiple names:

```
public class SayHelloArray {
  // Pass array of names to method
  static String sayHello(String[] names) {
    // Local variable to assemble greeting
    String greeting = "";
    for(String name : names) {
      greeting += "Hello, " + name + "\n";
    }
    return greeting;
  }

  public static void main(String[] args) {
    String[] userNames = {"Sophia", "Sofia", "Sophie"};
    // Pass variable userNames as parameter to sayHello()
    String greetingOutput = sayHello(userNames);
    // Pass variable greetingOutput to as parameter println()
    System.out.println(greetingOutput);
  }
}
```

The output should look like this:

© 2024 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

```
~/IntrotoJava$ java src/main/java/SayHelloArray.java
Hello, Sophia
Hello, Sofia
Hello, Sophie

~/IntrotoJava$
```

The signature for the sayHello() method, `sayHello(String[] names)`, includes square brackets to indicate that the parameter is an array of String values:

EXAMPLE

```
static String sayHello(String[] names) {
```

When the method is called in main(), though, note that the array is passed just by name without the square brackets:

EXAMPLE

```
String greetingOutput = sayHello(userNames);
```

Java methods can also take collections as parameters. Remember that collections can be generic:

```
import java.util.ArrayList;

public class SayHelloCollection {
  // Pass ArrayList of names to method
  // Remember that collections are generic. T is placeholder
  // for the data type
  static <T> String sayHello(ArrayList<T> names) {
    // Local variable to assemble greeting
    String greeting = "";
    // T is the data type
    for(T name : names) {
      greeting += "Hello, " + name + "\n";
    }
    return greeting;
  }

  public static void main(String[] args) {
    ArrayList<String> userNames = new ArrayList<>();
    userNames.add("Sophia");
    userNames.add("Sophie");
    userNames.add("Sophie");
    // Pass variable userNames as parameter to sayHello()
    String greetingOutput = sayHello(userNames);
    // Pass variable greetingOutput to as parameter println()
    System.out.println(greetingOutput);
  }
}
```

In this case, String is the most likely data type, but the method could work with the Character type (perhaps representing an initial):

```
import java.util.ArrayList;

public class SayHelloCollection {
  // Pass ArrayList of names to method
  // Remember that collections are generic. T is placeholder
  // for the data type
  static <T> String sayHello(ArrayList<T> names) {
    // Local variable to assemble greeting
    String greeting = "";
    // T is the data type
    for(T name : names) {
      greeting += "Hello, " + name + "\n";
    }
    return greeting;
  }

  public static void main(String[] args) {
    ArrayList<Character> userInitials = new ArrayList<>();
    userInitials.add('A');
    userInitials.add('B');
    userInitials.add('C');
    // Pass variable userInitials as parameter to sayHello()
    String greetingOutput = sayHello(userInitials);
    // Pass variable greetingOutput to as parameter println()
    System.out.println(greetingOutput);
  }
}
```

When passing a "plain" variable to a method, the code in the method works with a local copy of the data. The original value, in main() or wherever the method was called from, is not changed if the copy passed to the

© 2024 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

method is changed. As noted, parameters serve as local variables in the method to which they pass data. That means that the variables are only accessible in the block of code that makes up the body of the method. This is not exactly the case with arrays, though.

Changes made to an array in a method actually change the "original" array outside of the method, as seen below:

```
import java.util.Arrays;

class MethodChangeArray {
  // This method does not return anything but still
  // changes the order of items in the array passed
  // to it.
  public static void sortArray(int[] values) {
    Arrays.sort(values);
  }

  public static void main(String[] args) {
    int[] numbers = {5, 4, 3, 2, 1};
    // Display array contents before method call
    System.out.println("Original array:");
    System.out.println(Arrays.toString(numbers));
    // Call method
    sortArray(numbers);
    System.out.println("Array after method call:");
    // Display array contents after method call
    System.out.println(Arrays.toString(numbers));
  }
}
```

The results from running this program show that the array has been sorted outside of the method:

```
~/IntrotoJava$ java src/main/java/MethodChangeArray.java
Original array:
[5, 4, 3, 2, 1]
Array after method call:
[1, 2, 3, 4, 5]
~/IntrotoJava$
```

A similar situation can arise when passing a collection as a parameter.

Here is a version of the same code that uses an ArrayList collection rather than a plain array:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

class MethodChangeCollection {
  // This method does not retun anything but still
  // changes the order of items in the ArrayList passed
  // to it.
  public static void sortArrayList(ArrayList<Integer> values) {
    Collections.sort(values);
  }

  public static void main(String[] args) {
    // ArrayList collection of Integer values
    ArrayList<Integer> numbers = new ArrayList<>();
    numbers.add(5);
    numbers.add(4);
    numbers.add(3);
    numbers.add(2);
    numbers.add(1);
    // Display array contents before method call
    System.out.println("Original ArrayList:");
    System.out.println(numbers.toString());
    // Call method
    sortArrayList(numbers);
    System.out.println("ArrayList after method call:");
    // Display array contents after method call
    System.out.println(numbers.toString());
  }
}
```

Here is the output from this code:

```
~/IntrotoJava$ java src/main/java/MethodChangeCollection.java
Original ArrayList:
[5, 4, 3, 2, 1]
ArrayList after method call:
[1, 2, 3, 4, 5]
~/IntrotoJava$
```

In addition to passing multiple values via array or collection parameters, Java also has another mechanism for passing a variable number of values to a method using a single named parameter. The parameter's data type is

© 2024 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

followed (without a space) by 3 dots (...). The sample below shows how to design a `getAverage()` method that calculates the average of one or more double values passed via a parameter named `values`.

To access the values passed, the code in the method used an enhanced for loop (for-each loop) to process each in turn:

```
class VarargsAverage {
  // The ... indicates that the method can take a variable
  // number of values as its parameter (in this case, doubles)
  public static double getAverage(double... values) {
    double sum = 0;
    int count = 0;
    // Use an enhanced for loop to iterate through values
    for(double number : values) {
      sum += number;
      count++;
    }
    return sum / count;
  }

  public static void main(String[] args) {
    double avg1 = getAverage(1.2, 3.3);
    System.out.println("avg1 = " + avg1);
    double avg2 = getAverage(1.2, 3.3, 4.5, 5.1);
    System.out.println("avg2 = " + avg2);
  }
}
```

Running this code produces this output:

```
~/IntrotoJava$ java src/main/java/VarargsAverage.java
avg1 = 2.25
avg2 = 3.525
~/IntrotoJava$
```

The next section covers arguments passed when a method is called in greater detail. In this section of the lesson, you have learned about the parameters associated with methods that provide a means for passing data into a method. The names and data types for the parameters are set as part of the method's signature.

TERMS TO KNOW

Parameter (also called a Formal Parameter)
A kind of variable that is used as a channel to pass data to a method.

Variables with Local Scope (more commonly called Local Variables)
Variables that can only be accessed between their point of declaration and the end of the block of code.

---

2. Arguments

The parameters that a method takes are the channels which are used to pass information into a method. This allows the code in the method to do its work. The parameters are the mechanism to pass information.

KEY CONCEPT

Parameters are not the information itself. The actual data is passed as arguments when the method is called.

The **arguments** passed to a method must match the parameters specified in the method's signature. The order and data type would need to match as well. For instance, the first method we looked at, `sayHello()`, has the following signature:

EXAMPLE

```
sayHello(String name)
```

This means that when the sayHello() method is called, it has to have a String value passed in, representing the name of the person to be greeted.

In the sample program above, a String variable was declared with a value like this:

EXAMPLE

```
String userName = "Sophia";
```

And in the sample program above, the variable userName was then passed to the method like this:

EXAMPLE

```
String greeting = sayHello(userName);
```

Rather than using a variable, it would have also been possible to call the `sayHello()` method using a literal String value:

EXAMPLE

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

KEY CONCEPT

When a variable is used to pass an argument, the data type is not specified in the parentheses, though it needs to be part of the variable's declaration. This is in contrast to the parameter in the method signature that has to include the data type.

If a method takes an array or collection as a parameter, just the name of the array or collection is passed. This is done without square brackets or accessing a specific element, since the whole array or collection is being passed.

Note the version of the `sayHello()` method that takes an array with this signature:

EXAMPLE

```
sayHello(String[] names)
```

The method call looks like this, with just the name of the array in the parentheses:

EXAMPLE

```
String greetingOutput = sayHello(userNames);
```

Recall that the userNames array was declared like this:

EXAMPLE

```
String[] userNames = {"Sophia", "Sofia", "Sophie"};
```

THINK ABOUT IT

Review the version of the code with the `sayHello()` method that takes an `ArrayList` as its parameter. Note how the version using a collection passes the name of the collection (similar to how an array argument is passed by name).

TERM TO KNOW

Arguments
The actual variable or literal value passed when a method is called.

SUMMARY

In this lesson, you learned about **method parameters** and **arguments**. You also learned that they are used when determining needed values. Finally, you looked into method parameters and arguments in greater detail.

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

Arguments	The actual variable or literal value passed when a method is called.
Parameter (also called a Formal Parameter)	A kind of variable that is used as a channel to pass data to a method.
Variables with Local Scope (more commonly called Local Variables)	Variables that can only be accessed between their point of declaration and the end of the block of code.

© 2024 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.