Object Oriented Programming – Java 1

Lecture 3

Methods

Dr. Obuhuma James

Description

This topic aims at exploring methods as units of execution that can be created and called to perform a given task in Java. Concepts for both void and value-returning methods are covered. Additionally, the process of calling methods with or without parameters is described. This relates to passing parameters to methods. Finally, the scope of variables with respect to their accessibility and usage within a Java program is also covered.

Learning Outcomes

By the end of this topic, you will be able to:

- Understand the concept of methods as used in programming.
- Create methods for any given programming task.
- Execute methods requiring and not requiring parameters.

Overview

The last two topics have introduced Java programming using simple statements lamped within one program code. This topic will dive into a different approach that allows grouping of statements into methods that can then be executed as single units. Methods embrace the concept of modularity in programming, where tasks are grouped and executed independently. Such an approach makes troubleshooting for errors much easier. In addition, methods facilitate code reuse in programming. It is through methods that the concept of variable scopes their usage become more real.

Up to this point in our course, we have already covered Java classes that contain one method called main() method. The main() method executes automatically when a program runs. The main() method can execute additional methods that can also execute others [1]. A Java class can contain an unlimited number of methods, each of which can be called an unlimited number of times [1]. To execute a method, you invoke or call it as will be seen later in the topic.

Methods

A method is a unit of execution that allows grouping of statements into one single unit meant to perform a given task. In other words, methods are program modules that contain a series of statements that carry out a given task [1]. Methods become handy in programming due to the following reasons:

1. They enable better organization of solutions to computational problems that in return separates large problems into smaller and more manageable parts.
2. They embrace code reusability such that once a method has been created for a given task, it can be reused as many times as needed within the same program or even in other programs. This in return leads to creation of useful shareable libraries.
3. Methods also facilitate collaboration in authorship of large programming projects where large projects can be split into smaller tasks allocated to individual programmers working on them independently. The programmers could later collaborate to assemble the final program.
4. Finally, methods make troubleshooting program code much easier than if the program code is delivered as one large program.

Methods can be either predefined or user-defined. Most of the predefined methods are mathematical methods that include abs(x), log(x), exp(x), max(x, y), min(x, y) among others [2]. On the other hand, user-defined methods can be broadly categories as void methods or value-returning methods.

1. Void Methods

Void methods are said to return nothing to the caller. Thus, their return type is normally set to void.

Defining Void Methods

Syntax

```
accessspecifier statickeyword void methodname(parameter-list){
        statement(s) to be executed;
}
```

Methods must have a name that identifies them in the program. The name borrows similar naming convention for identifiers, as covered in earlier topics. The list of parameters in a method is optional. Such a list is included or excluded depending on whether the method requires external values to be supplied as parameters for it to execute or not. This will be covered in subsequent subsections. In cases where no parameter is required, then the brackets following the method name must remain, but left empty. Void methods do not specify a return type for the method, thus, the term void.

Example

The following is an example of a void method that computes the product of two numbers stored in variables x and y. The method display the product without returning anything to its caller.

```
public static void product(){
        int x = 4;
        int y = 3;
        System.out.print(x * y);
}
```

Executing Void Methods

To execute a method, a call to the method must be supplied somewhere within the program. This could be in another method that has a call to the main() method or directly called in the main() method.

Syntax

methodname(argument-list);

The list of arguments is supplied in the method call if and only if the method being called to execute has parameters in its definition. Otherwise, in cases where no parameter is to be passed to the method, then the brackets must remain, but left empty.

Example

The following is an example of a call to the product method created in the previous subsection to compute the product of two numbers stored in variables x and y.

product();

The method code is basically executed at the point where it is called. Printing of any output from the method is done within the body of the method, since it is a void method.

Figure 1 shows the complete program that demonstrates the concept of void methods. The output is 12 as shown in the output window. This is as a result of the method's call to execute in the main() method.

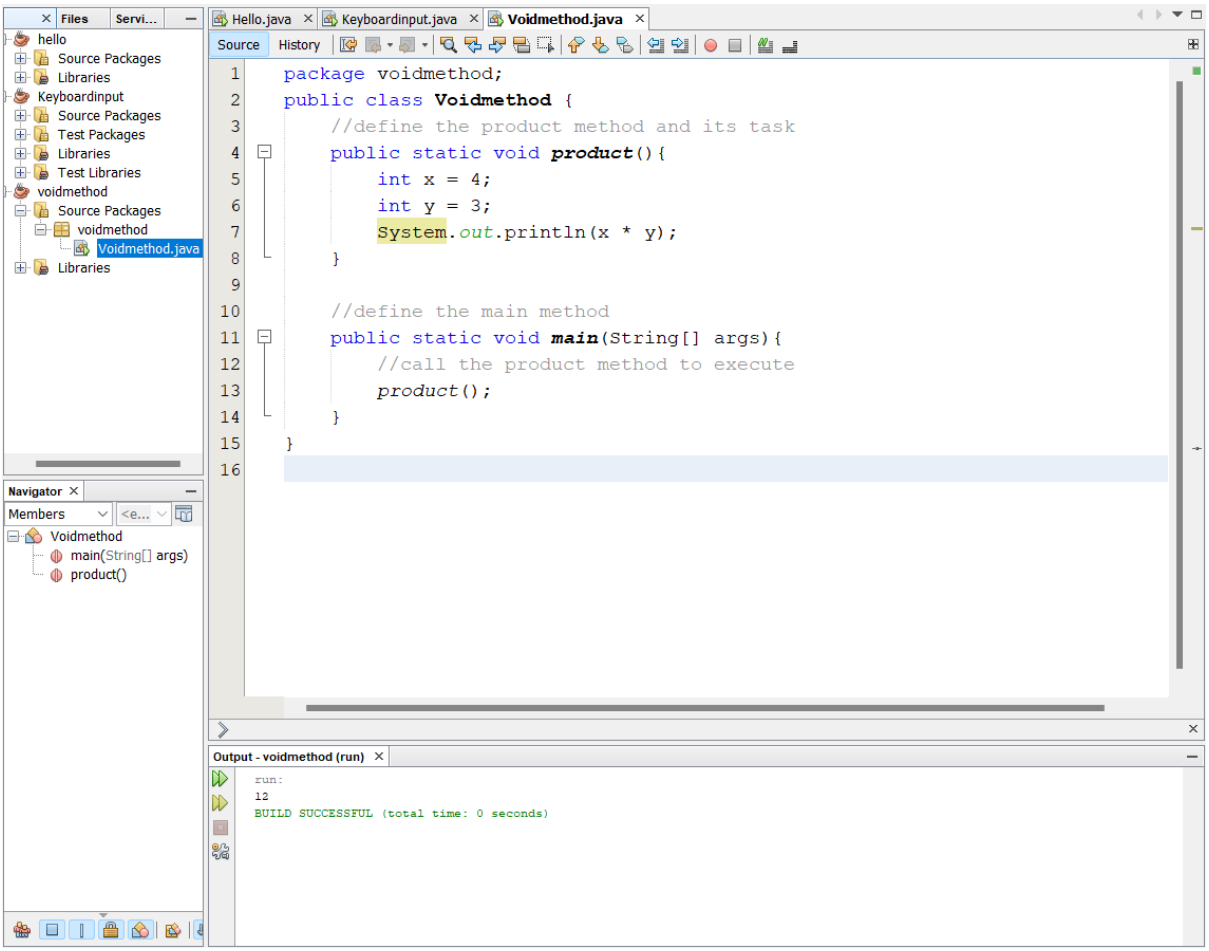

Figure 1. Void Method Example

2. Value-Returning Methods

Methods can also be defined in such a manner that makes them only execute whatever task they are meant to do, after which, they return the results to the caller. In such cases, the method does not contain any statements for printing out anything, instead, it only returns values/results to the caller. Methods of this kind must have a return statement that dictates what is to be returned to the caller.

Defining Value-Returning Methods

Syntax

```
accessspecifier statickeyword returntype methodname(parameter-list){
        statement(s) to be executed;
        return value;
}
```

The format for a value-returning method is similar to that of a void method except for a minor change and addition.

- A return type is a data type that specifies the type of results that the method is expected to return to its caller.
- The return statement at the end of the method's body specifies the value/result to be returned to the caller.

Example

Consider the previous product() method example. Suppose we want the method to now return the product to the caller rather than print it out within its body, we will modify the method as follows.

```
public static int product(){
        int x = 4;
        int y = 3;
        return x * y;
}
```

By introducing int as the return type and replacing the System.out.print() statement with a return statement, the method will strictly return results of multiplying 4 by 3 to the caller. This transforms the method into a value-returning method.

Executing Value-Returning Methods

To execute a value-returning method is more or less similar to the process of executing a void method. The only difference is that we should be cognizant of the fact that the method being called will only return results. Hence, it is at the point of calling onwards that determines what is to be done on the returned results. For instance, to printout the results on the console, the caller should subject the method call to the System.out.print() statement.

Syntax

System.out.print(methodname(argument-list));

Example

The following is an example of how to call and printout the results of the value-returning product() method created in the previous subsection to compute the product of two numbers stored in variables x and y.

System.out.print(product());

Notice the fact that the method call has been subjected to the System.out.print() statement. This facilitates display of value 12 that is returned by the method. In case the System.out.print() statement is not added to the method call, there will be no errors at all, however, nothing will be displayed on the console. This is because the method will have executed successfully following the call, but, since the results returned have not been printed, nothing appears.

Value-returning methods come with their own advantages with the main one being the fact that the method gives the programmer the liberty to decide what to do with the results returned by the method. Two main ways to use such results:

1. Print them as they are by directly passing the method call to the System.out.print() statement, as previously demonstrated. Figure 2 shows the full program that demonstrates the concept of value-returning methods of this nature.

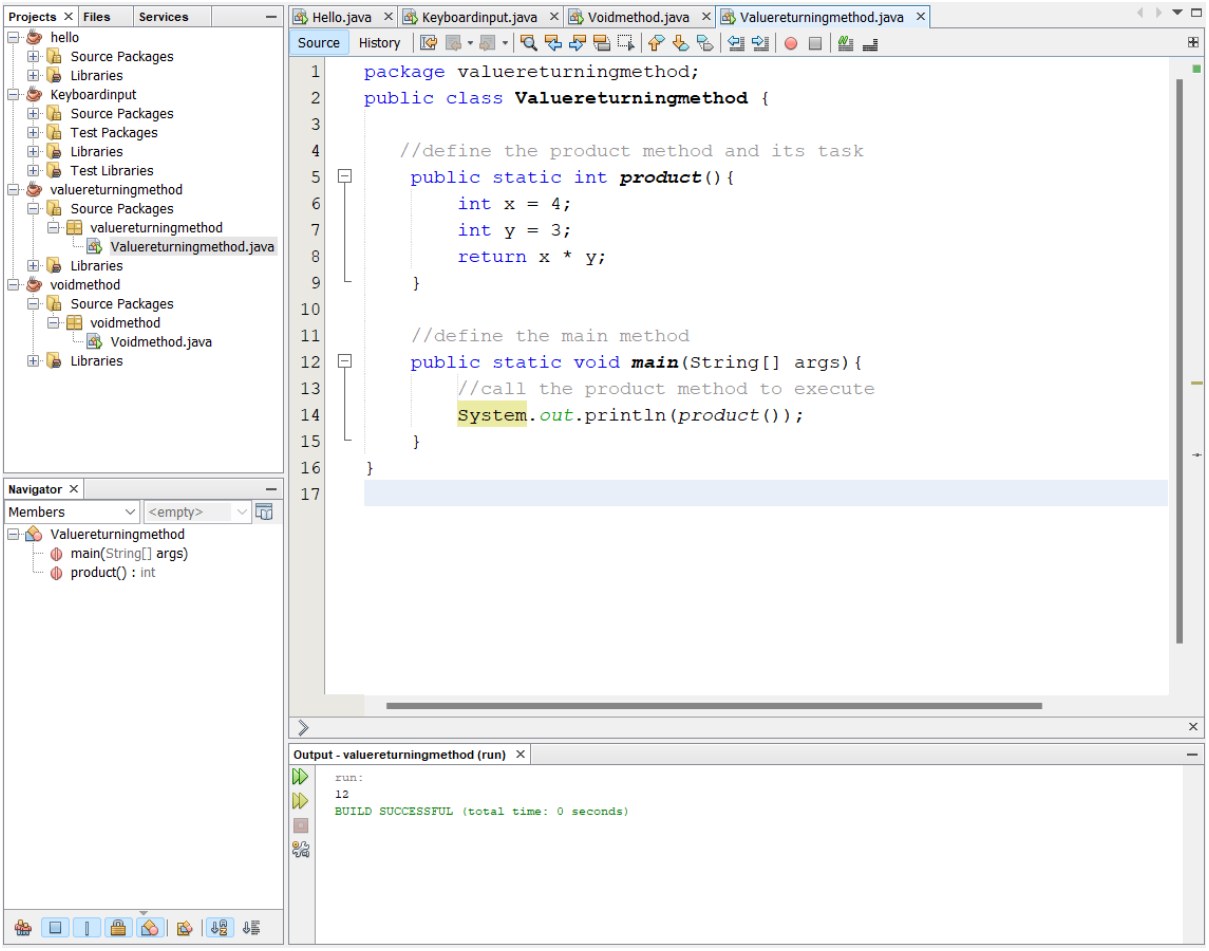

Figure 2. Value-returning Method Example 1

2. Pass the results to other methods or expressions in cases where further actions must be performed to the returned results rather than just printing them as they are. Consider the program in Figure 3 that implements this by computing total deductions from an employee's monthly salary.

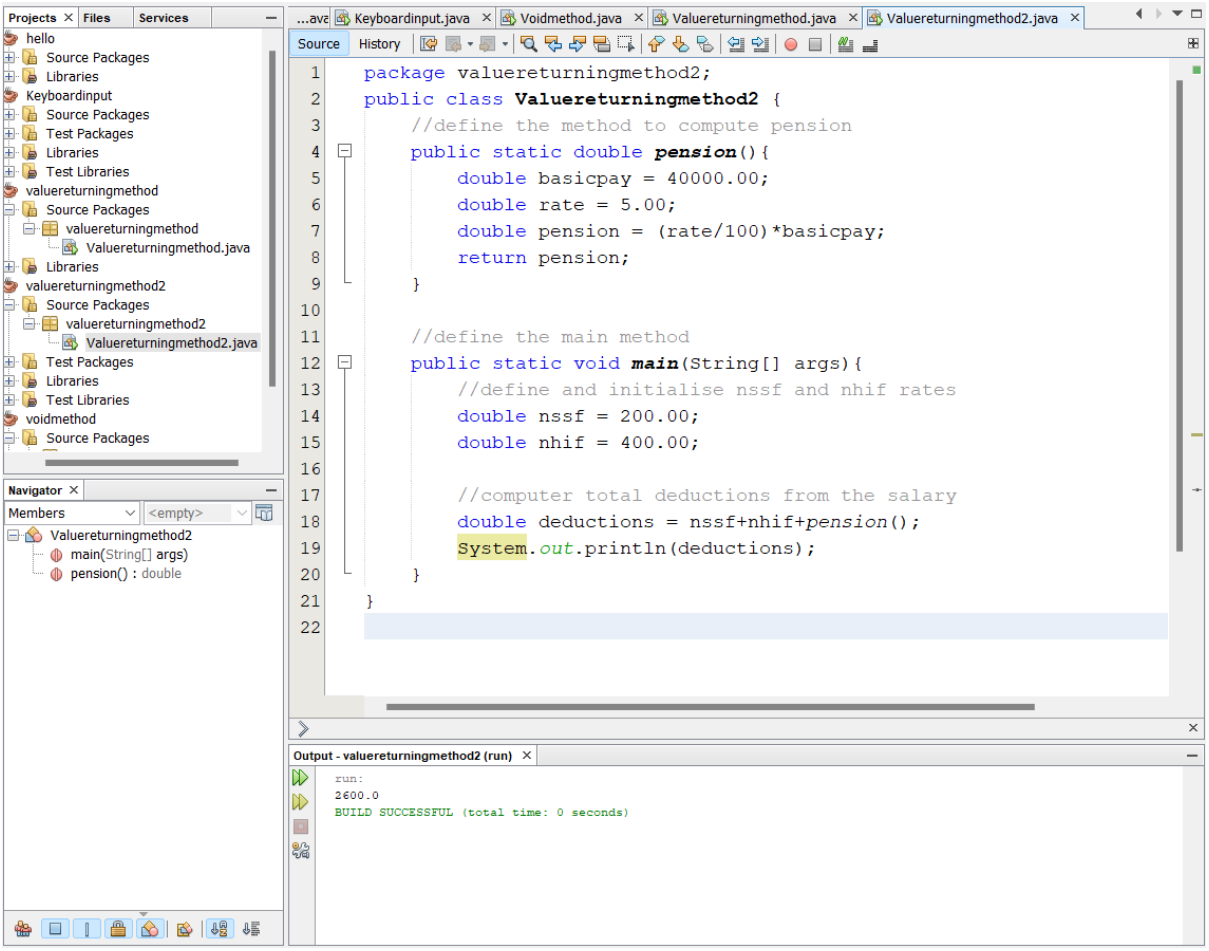

Figure 3. Value-returning Method Example 2

The program uses a value-returning method to compute pension, which is 5% of the basic pay. The pension together with the National Social Security Fund (NSSF) and National Hospital Insurance Fund (NHIF) are then summed together to generate total deductions. Notice the fact that the value returned by the pension() method has been directly passed to the expression computing deductions before a separate System.out.print() statement is used to display the total deductions. This is a good demonstration of the flexibility in the use of value-returning methods.

Methods with Parameters

As earlier mentioned, methods can also be defined in such a manner that they require parameters for execution. In such cases, the parameters must be passed as arguments at the point of calling the methods. The number of arguments passed must match the number of expected parameters in the method's definition. There are two ways to pass parameters to a method, namely, pass by value and pass by reference [1].

Passing by value creates a local copy of the variable that is used by the method during execution [1]. As a result, any changes made to the parameter's value within the method are lost when control is passed from the method back to the program. On the other hand, passing by reference makes the actual variable to be used within the method, which makes changes made to the variable at the point of method execution to remain as control is passed back to the program. In such cases, the address is passed to the method and not the value [1]

Example

Consider the modified initial program that was computing the product of two numbers, such that this time round the values for x and y are passed through a method call. The example as shown in Figure 4 demonstrates the concept of passing by value.

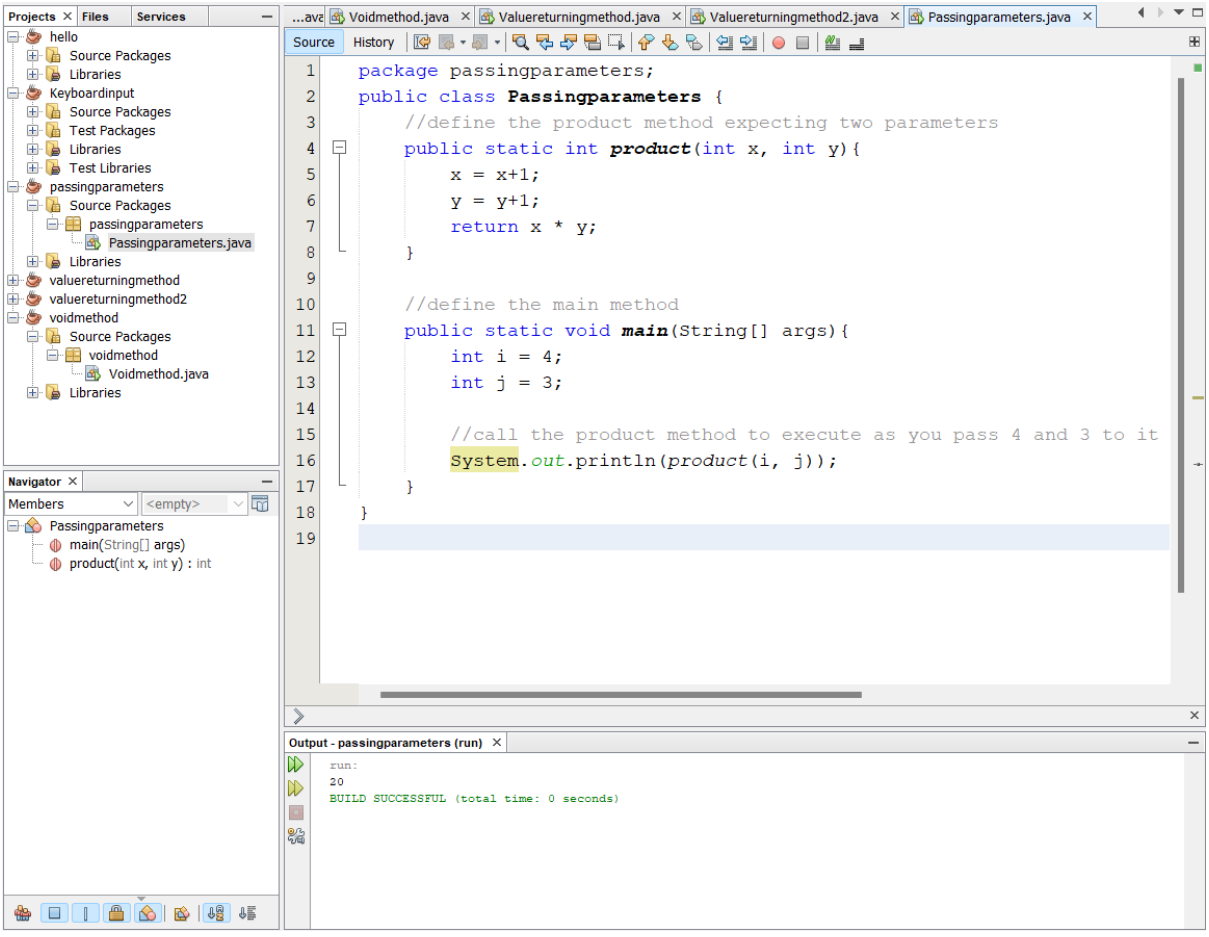

Figure 4. Passing Parameters to Methods

Variable Scope

It is important to understand the impact that blocks and methods have on your variables [1] to avoid unnecessary errors in programs. In any given program, variables with the same names represent different memory locations when they are declared within different scopes [1]. It is worth noting that a variable ceases to exist, or gets out of scope, at the end of the block in which it is declared [1]. Thus,

1. A similar variable name cannot be declared more than once within a block, even if the block contains other blocks.
2. A given class's instance variables override any locally declared variables with the same names that are declared within the class's methods.

Accessor and Mutator Methods

Despite the broad categorization of methods as either void or value-returning, methods in Java can also be categorized as either mutator or accessor methods. Mutator methods, also known as setters, are used to set or change values [1 – 2]. Accessor methods, also known as getters, are used to retrieve values from mutators [1 – 2]. The concept of accessors and mutator will be demonstrated in the next topic after the introduction of programs with more than one class and the concept of objects.

Summary

The topic has covered methods and how they can be used to perform any given task. Two perspectives of defining and using methods have been explored, namely, value-returning and void methods. Furthermore, the topic has clearly demonstrated the concept of passing parameters, with an example focused on passing by value for cases where methods require parameters for execution. Finally, the scope of variables with respect to how and where they can be accessed and used has also been discussed in brief.

Check Points

1. Discuss the reason behind the existence of methods in programming.
2. Differentiate between value-returning and void methods.
3. Describe the approach used to execute methods requiring parameters and those not requiring parameters.

4. Differentiate between passing by value and passing by reference as two mechanisms used to pass parameters to methods.
5. Differentiate between mutator and accessor methods.
6. Using an appropriate example describe what is meant by variable scopes.

Core Textbooks

1. Joyce Farrell, Java Programming, 7th Edition. Course Technology, Cengage Learning, 2014, ISBN-13 978-1-285-08195-3.
2. Malik, Davender S. JavaTM Programming: From Problem Analysis to Program Design, International Edition, 5th Edition, Cengage Learning.

Other Resources

3. Daniel Liang, Y. "Introduction to Java Programming, Comprehensive." (2011).
4. Malik, Davender S. JavaTM Programming: From Problem Analysis to Program Design, International Edition, 4th Edition, Cengage Learning, 2011.
5. Shelly, Gary B., et al. Java programming: comprehensive concepts and techniques. Cengage Learning, 2012.

References

[1] Farrell, J., Java Programming, 7th Edition. Course Technology, Cengage Learning, 2014, ISBN-13 978-1-285-08195-3.

[2] Malik, D. S., JavaTM Programming: From Problem Analysis to Program Design, International Edition, 5th Edition, Cengage Learning.

[3] Sebester, R. W., Concepts of Programming Languages, 12th Edition, Pearson, 2018, ISBN 0-321-49362-1.