Object Oriented Programming – Java 1

Lecture 5

Control Structures: Decision Making

Dr. Obuhuma James

Description

This topic aims at looking at a different perspective of execution of statements in a program code. So far, the form of execution covered has been purely sequential, where statements in a program code are executed sequentially in the order in which they appear. Introduction of control structures shift the order of execution to be purely user-defined as opposed to linear. There are two types of control structures, namely, decision-making and looping. This topic will focus on decision-making while the next topic will dwell on looping. The various decision-making options will be covered with examples in each case.

Learning Outcomes

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

- Understand the reasons behind control structures as applied in computer programming.
- Comprehend on the concept of decision-making control structures.
- Demonstrate the various options used to implement decision-making in programming.

Overview of Control Structures

Statements in a computer program can be processed in three different fashions, namely, sequential, branching, or looping [2]. The form of execution of statements covered so far has been purely sequential where statements are executed in the order in which they appear in the program, i.e., a top-down manner. In the sequential approach, one step follows another unconditionally, with no branching or skipping of steps [1]. This topic introduces the concept of branching that allow execution to deviate from sequential or linear fashion to an order that is based on the results of a decision, selection or choice [2]. Branching or decision-making is somewhat universal since the logic cuts across relatively all programming languages. The next topic will continue on the concept of control structures by exploring looping which is a concept that alters the flow of program execution through repetition or iteration of program statements [2]. Figure 1 summarises the three kinds of execution.

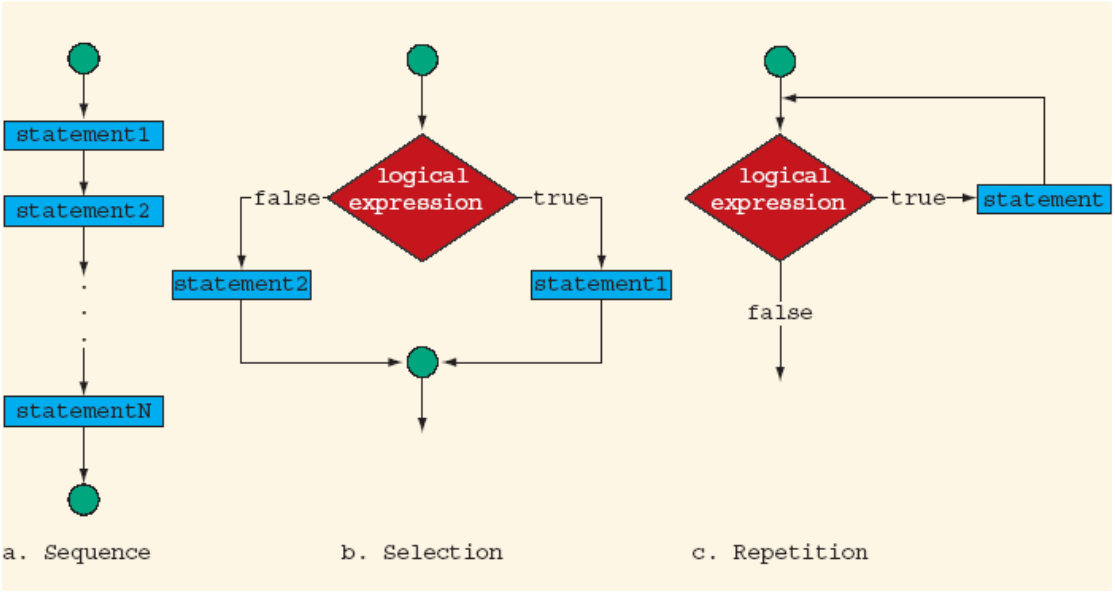

Figure 1. Types of Execution [2]

Decision-making

Decision-making is a control structure that allows for the order of execution to take one of two options available based on the outcome of a given conditional expression. It is also known as branching or selection. The conditional expression returns a Boolean value that can be either TRUE or FALSE as a determinant of the direction to be taken as shown in Figure 2.

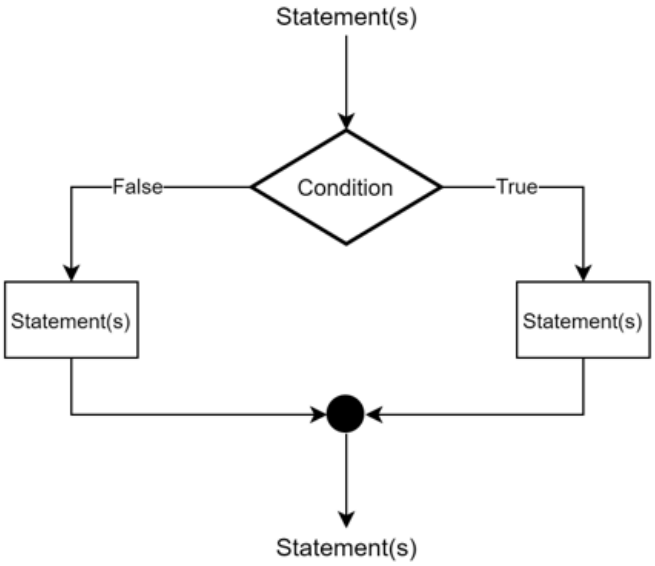

Figure 2. The Concept of Decision-making Control Structure

There are three statements that can be used to implement decision-making in Java, just like in other programming languages. These are the if statement, the if … else statement and the switch statement.

The if Statement

The if statement is the most basic and simplest decision-making statement [1]. The if statement contains a condition that evaluates to a TRUE or FALSE Boolean value. In case the condition returns a TRUE, then the statements in the if segment are executed. Otherwise, if a FALSE value is returned, then the program does not get into the if segment, it instead proceeds execution of any other statements coming after the if statement segment.

Syntax

```
if(condition){
        Statement(s) to be executed of condition is TRUE;
}
```

Example

Consider the following simple example of a Java program that checks whether the temperature value provided is greater or equal to 20 for it to execute or fail to execute the if segment of the program.

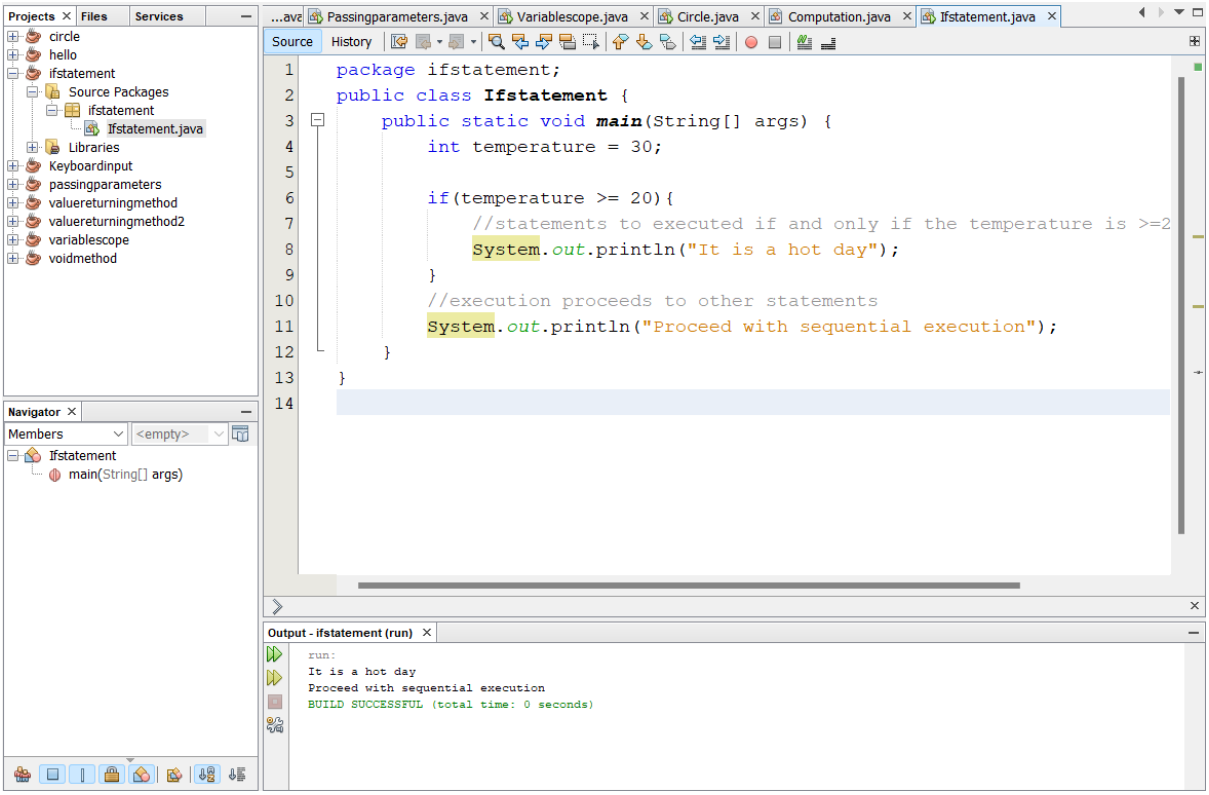

Figure 3. Example of the if Statement

Since the set temperature value 30 is greater than 20, the condition returns a TRUE. This makes the execution to jump into the if statement block and execute the statement that displays “It is a hot day”. After which, execution proceeds to sequentially execute any other statements coming after the if statement segment. If the set temperature could have been less than 20, the output could have omitted the statement “It is a hot day”.

The if … else Statement

The if … else statement extends the standard if statement by including an else part. The else segment comes immediately after the if segment, such that it only gets executed if and only if the condition returns a FALSE value.

Syntax

```
if(condition){
        Statement(s) to be executed if condition is TRUE;
}else{
        Statement(s) to be executed if condition is FALSE;
}
```

Example

Consider the enhanced example of the Java program that checks whether the temperature value provided is greater or equal to 20. This time round, the program will either jump into the if segment or else segment depending on the outcome of the condition.

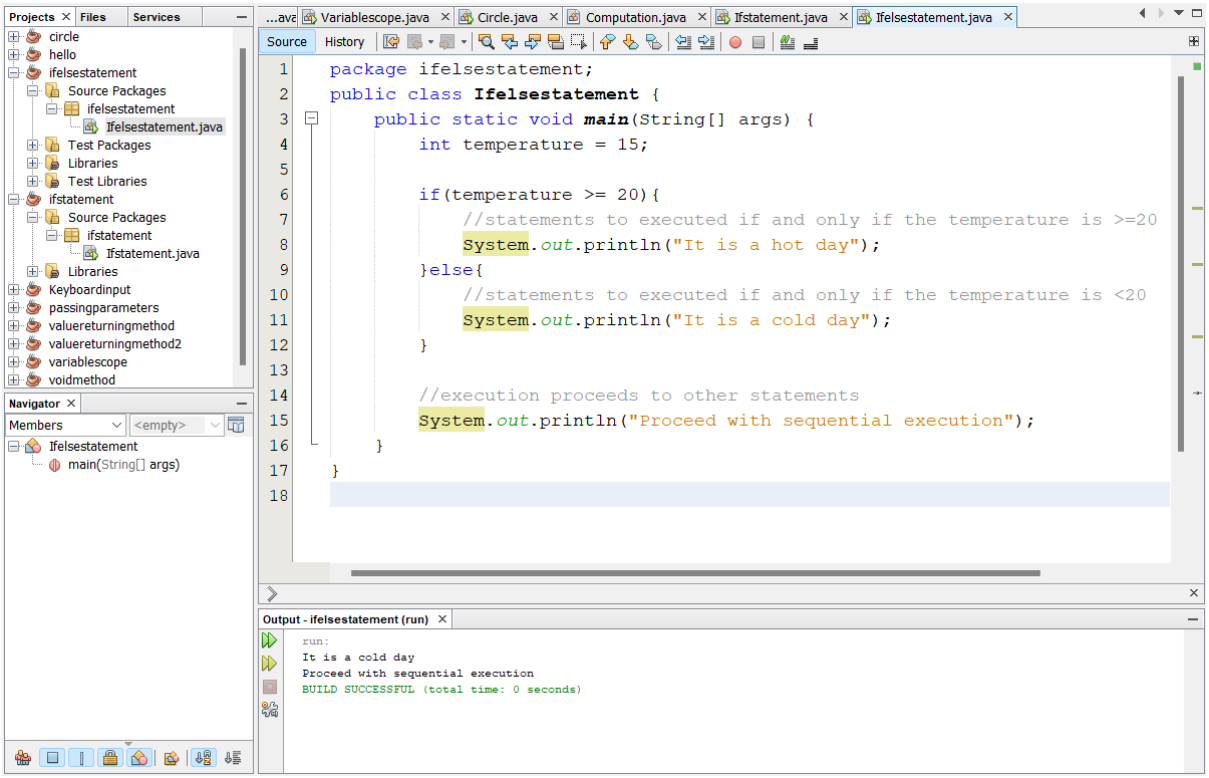

Figure 4. Example of the if … else Statement

Since the set temperature value 15 is less than 20, the condition returns a FALSE. This makes the execution to jump into the else part of the if … else statement and execute the statement that displays “It is a cold day”. After which, the program proceeds to sequentially execute any other statements coming after the if and the else segments. If the set temperature could have been 20 or more, the output could have been as explained in the previous if statement example.

The Nested if … else Statement

The if … else statement allows nesting, where several if … else statements are stacked together to test several conditions. Such an implementation is commonly referred to as the nested if … else.

Syntax

```
if(condition_1){
        Statement(s) to be executed if current condition is TRUE;
}else if(condition_2){
        Statement(s) to be executed if current condition is TRUE;
}else if(condition_3){
        Statement(s) to be executed if current condition is TRUE;
}
        …

else if(condition_n){
        Statement(s) to be executed if current condition is TRUE;
}else{
        Statement(s) to be executed if no condition is TRUE;
}
```

Example

Consider a Java program that checks a numeric value matching a day of week then displays the correct day of week, such that:

1 – Sunday

2 – Monday

3 – Tuesday

4 – Wednesday

5 – Thursday

6 – Friday

7 – Saturday

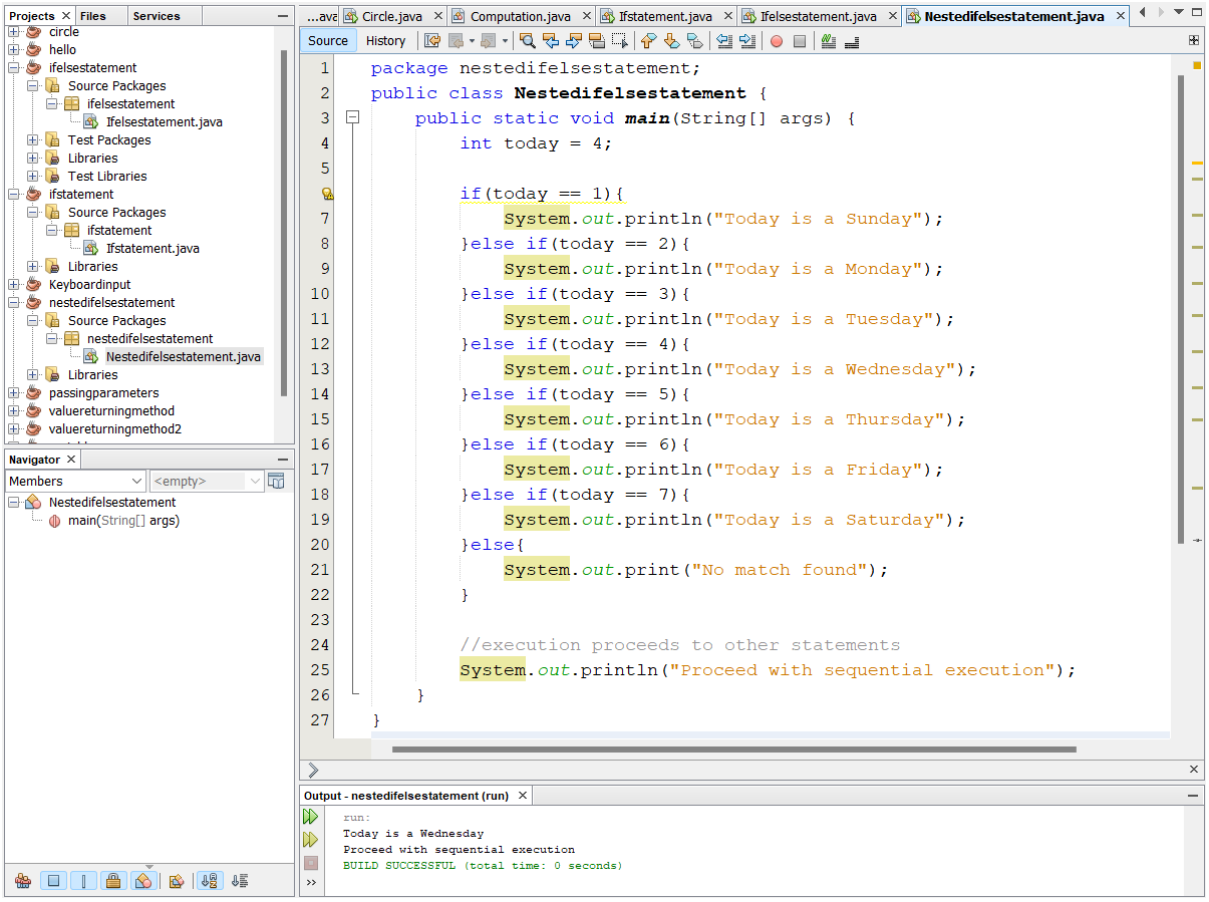

Figure 4. Example of the Nested if … else Statement

Since the variable, today, has been set to value 4, only condition 4 returns a TRUE. This makes the program execution to jump into that segment and execute the statement that displays "Today is a Wednesday". After which, the program proceeds to sequentially execute any other statements coming after the nested if … else segment. If none of the conditions could have returned a TRUE, then the else part of the nested if … else segment could have been executed. Note that the else part comes at the very end of the nested if … else statement to take care of such scenarios.

It is worth pointing out that the nested if … else statement works in a similar manner to that of a switch statement. Thus, the next subsection on the switch statement will hence use a similar example to show the relationship between the two approaches.

The switch Statement

The switch statement is best suited for scenarios with many options requiring decision-making. It starts with a switch kind of function that determines the value of focus. Such a

values may be supplied directly or may be a result of an expression. The value can be of different data types including String, character, or numeric value. The switch body contains as many case segments as the number of possible options to be tested. Each case has a different value tested against the value being sought for. A break statement exists in each case segment to allow the execution to get out of the switch statement upon successful execution of all statements in the matching case segment. At the very end of the cases is a default segment that is executed if and only if no matching case is found. It is very important to ensure that the break statement is present in each case segment. Otherwise, the default segment will be executed in addition to the matching case since the execution will still be in the switch statement [1].

Syntax

```
switch(expression/value){
        case value_1:{
                Statement(s) to be executed if the value matches;
                break;
        }
        case value_2:{
                Statement(s) to be executed if the value matches;
                break;
        }
        case value_3:{
                Statement(s) to be executed if the value matches;
                break;
        }
                …
        case value_n:{
                Statement(s) to be executed if the value matches;
                break;
        }
        default:{
                Statement(s) to be executed if the value matches;
        }
}
```

Example

The following is a similar implementation for a program matching numeric values to days of week using the switch statement.

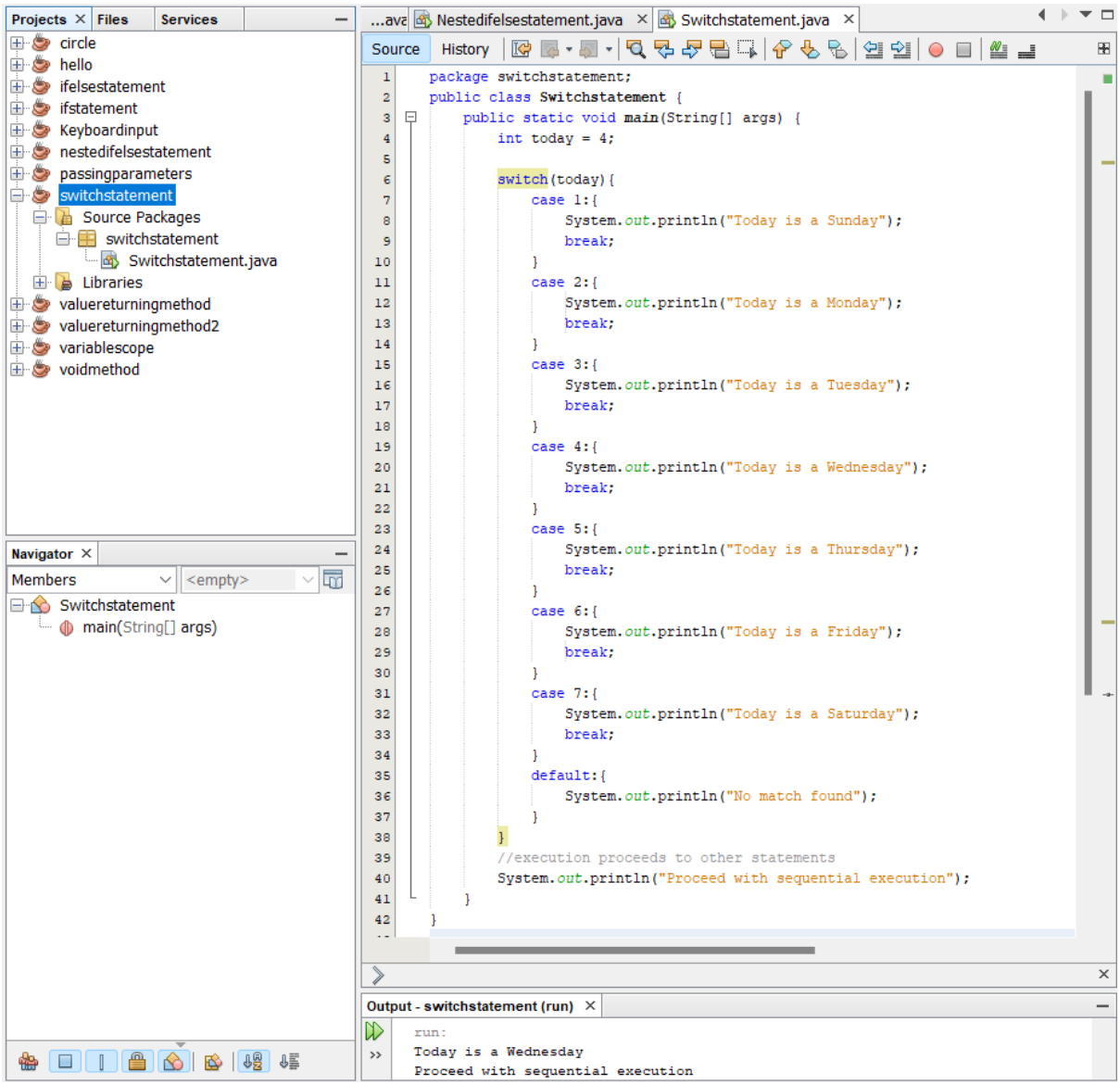

Figure 5. Example of the Switch Statement

Since the variable, today, has been set to value 4, only the case requiring value 4 matches. This makes the program to jump into that segment and execute the statement that displays “Today is a Wednesday”. After which, the program breaks from the segment allowing the program to proceed sequentially by executing any other statements coming after the switch statement segment. If none of the cases could have been found matching, then the default segment could have been executed. Note that the default part comes at the very end to take care of such scenarios.

Summary

In this topic, the concept of control structures has been introduced. This has broken the monotony of sequential execution by exploring decision-making as the second type of execution of program statements. Three statements that implement decision-making have been covered. These are the if … statement, the if … else statement, and the switch statement. The concept of nesting of if … else statements has also been covered as another version of implementing the switch statement. The next topic will focus on looping as the third and last form of execution.

Check Points

1. Discuss the term control structures as used in computer programming.
2. Describe the three types of execution of statements as used to computer programming.
3. Demonstrate the concept of the if … statement as a decision-making control structure.
4. Demonstrate the concept of the if … else statement as a decision-making control structure.
5. Using an example, explain the relationship between the nested if … else statement and the switch statement.

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.