Object Oriented Programming – Java 1

Lecture 12

File Input and Output

Dr. Obuhuma James

Description

This is the final topic in this course. The topic aims at exploring how file input and output can be performed in Object-oriented Programming languages, particularly in Java. This may not necessarily be a pure object-oriented programming concept. However, it is a major aspect of programming. The process of reading from and writing into a text file will be explored. This will form a better foundation to an advanced course on Java Programming that will cover Java to database connectivity.

Learning Outcomes

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

- Describe the concept of file input and output as applied in computer programming.
- Demonstrate how to read from a text using a programming language.
- Demonstrate how to write into a text using a programming language.

Overview of File Input and Output

Part of computer programming entails being able to store data in some form of files. This may not necessarily be a pure object-oriented concept. It is however a major aspect of programming worth exploring. A file in a computer may be perceived as a collection of information stored on a nonvolatile computing system device [1]. Nonvolatility in this case means that the information is somehow permanent such that it is not lost when the device holding it loses power [1]. Some examples of permanent storage devices include hard disks, USB drives, compact disks, among others.

According to Farrell [1], files can be categorized by the way they store data. Thus, based on this, some of the file categories include text files, binary files, data files, program or application files [1].

- Text file – a computer file structured as a sequence of lines of electronic text. Normally saved with a .txt file extension in the Windows Operating system.
- Binary file – a computer file that contains non-textual data. Binary file content is usually in a format that can only be interpreted by specific programs or understandable to specific hardware processors.

- Data file – a computer file that contains data only meant for reading or viewing and not for execution. Such data may be used as input or output for a computer of application. Normally saved with a .dat file extension in the Windows Operating system.
- Program or application file – a computer file that stores code to run a program. Normally saved with a .exe file extension in the Windows Operating system. Such a file is commonly referred to as an executable file.

Regardless of the type of file, a number of operations can be performed. These includes opening a file, writing into a file, reading from a file, closing a file, and deleting a file. Above all, a determination of whether the file exists and the path to the file location is also critical. This topic will focus on the process of reading data from a text file and writing information into a text file using the Java programming language.

Reading Data from a File

Reading data from a text file using Java makes use of the Scanner class and its object that was covered in our earlier topics. In our earlier topics, the Scanner class aided in the definition of a Scanner object. This was done hand in hand with the System.in object targeting the standard input device, in this case the keyboard. In this topic, where the aim is to read from a text file, the FileReader class is used to focus of a text file as the input source. The following steps summarises the process of reading from a file:

1. Import necessary classes from the java.util and java.io packages.

   Syntax

   import java.util.Scanner;

   import java.io.FileReader;

2. Create an object of the Scanner class with the FileReader class specifying the source file.

   Syntax

   Scanner objectname = new Scanner(new FileReader(filename_and_path));

3. Use the Scanner object to read data from the source file in a similar fashion that input data from the standard input device was done in our earlier topic. In this case, the next(), nextInt(), nextLine, among other methods are put in use.

   Syntax

   objectname.methodname();

4. Close the file using the close() method.

   Syntax

   objectname.close();

Reading from a Text File Example

Consider a text file called data.txt in Figure 1 consisting of a line of sample data where the data represents first name, last name, age and salary respectively.

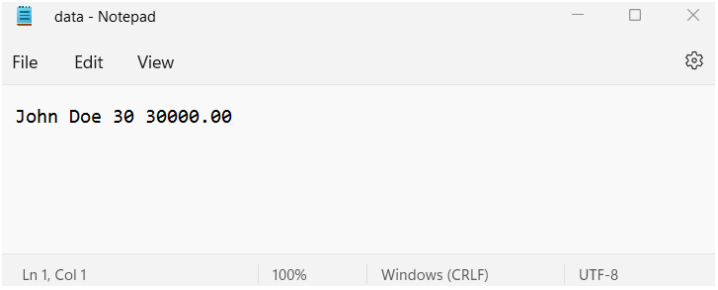

Figure 1. A Text File with Sample Data

Figure 2 shows a Java program that demonstrates how the data in the data.txt file could be read and displayed on the console. Notice the output window that show the data having been successfully read and displayed on the console.

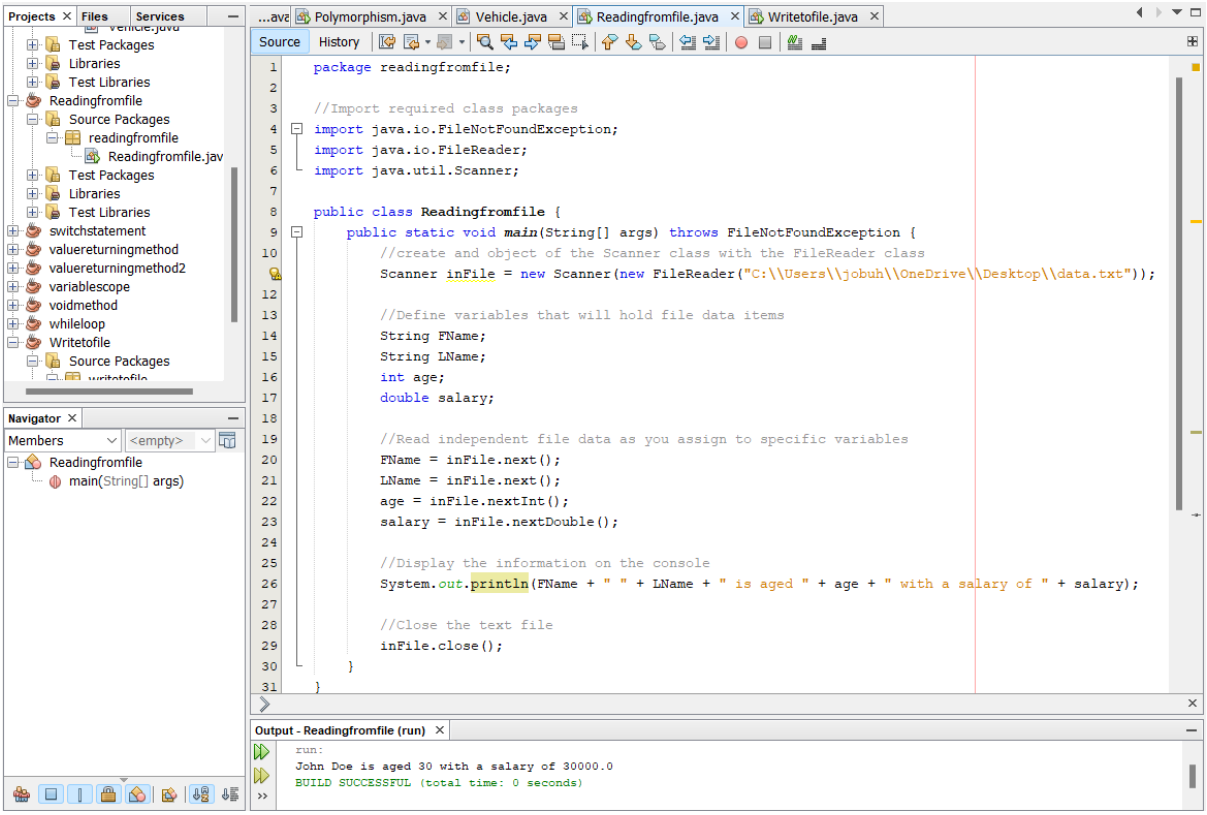

Figure 2. A Java Program that Read the Text File Sample Data

Writing Information into a File

Writing to a text file using Java makes use of the PrintWriter class. It is a fairly straight forward process. The following steps summarises the process of writing to a file

1. Import the necessary class from the java.io package.

   Syntax

   import java.io.PrintWriter;

2. Create an object of the PrintWriter class specifying the destination file.

   Syntax

   PrintWriter objectname = new PrintWriter(filename_and_path);

3. Use the print() or println() methods to write the data into the destination in a similar fashion that data was written on the console in our earlier topic.

   Syntax

   objectname.print();

4. Close the file using the close() method.

   Syntax

   objectname.close();

Writing to a Text File Example

Consider an empty text file called data.txt as shown in Figure 3.

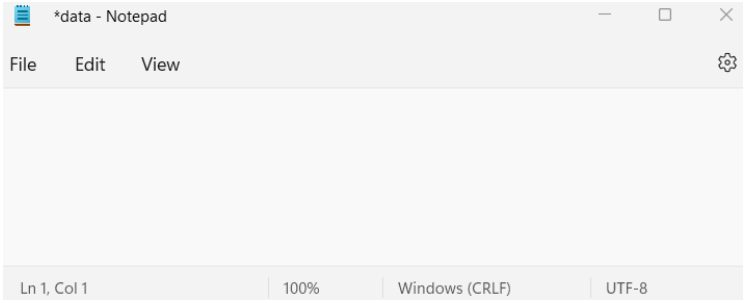

Figure 3. An Empty Text File

Figure 4 shows a Java program that demonstrates how data could be written in the data.txt file. Notice the output window that show that the program executed (run) successfully.

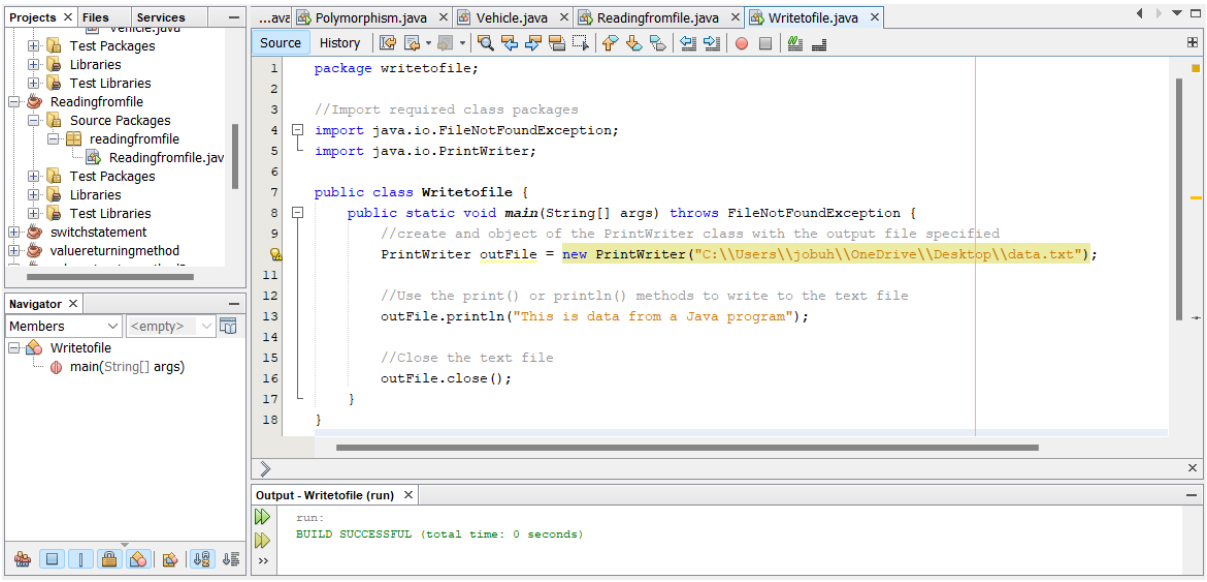

Figure 4. A Java Program that Writes into a Text File

Figure 5 shows the initially text file, this time round with the data that was written into it following successful program execution.

Figure 5. A Text File with Data Following Program Execution

It is worth noting that the concept of exception handling has been applied in both examples by appending the "throws FileNotFoundException" statement at the end of the main() method definition. This is due to the fact that an exception relating to a file not being found is expected. Note that the try … catch … finally approach could still have applied in this context.

Summary

The topic has discussed the various types of computer files. Regardless of the type of file, a number of operations can be performed on files, including, opening a file, writing into a file, reading from a file, closing a file, and deleting a file. Thus, the topic has demonstrated the process of reading from a text file, writing into a text file, and closing a text file. The concept of exception handling has been applied, but in a slightly different manner.

Check Points

1. Describe the difference among the various types of computer files.
2. Outline the common operations that can be performed on computer files.
3. Using an example, demonstrate the process of reading from a data file.
4. Using an example, demonstrate the process of writing into a data file.
5. Describe how the concept of exception handling applied to the process of reading from and writing into computer files using Java.

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.