



WORKING WITH FILES AND INPUT/OUTPUT

OBJECT ORIENTED PROGRAMMING I

Sercan Külcü | Object Oriented Programming I | 10.01.2023

Contents

Introduction	2
What are Files and Input/Output in Java?.....	3
How to Create and Read Files in Java?	4
Best Practices Implementing Input/Output Operations	6
Conclusion	7

Introduction

Java is a versatile programming language that can handle various types of input and output operations. In this chapter, we will explore how to work with files and input/output (I/O) in Java, including what they are, how to create and read files, and best practices for implementing input/output operations in your code.

What are Files and Input/Output in Java?

In Java, files are used to store data permanently on a disk or other storage devices. The input/output (I/O) operations, on the other hand, refer to the process of reading data from and writing data to different sources, including files, databases, network sockets, and standard input/output devices.

How to Create and Read Files in Java?

Creating and reading files is a fundamental operation in Java. Here's how you can create and read files in Java:

Creating a file: To create a file in Java, you can use the File class. First, create an instance of the File class and pass the file path as a parameter to the constructor. Then, use the createNewFile() method to create a new file.

Writing to a file: To write data to a file, you can use the FileWriter class. First, create an instance of the FileWriter class and pass the file path as a parameter to the constructor. Then, use the write() method to write data to the file.

Reading from a file: To read data from a file, you can use the FileReader class. First, create an instance of the FileReader class and pass the file path as a parameter to the constructor. Then, use the read() method to read data from the file.

Here is an example of how to create and read a file in Java:

```
import java.io.*;

public class FileExample {
    public static void main(String[] args) {
        try {
            File file = new File("example.txt");
            if (file.createNewFile()) {
                System.out.println("File created: " +
file.getName());
            } else {
                System.out.println("File already exists.");
            }
            FileWriter writer = new FileWriter(file);
            writer.write("Hello, world!");
        }
    }
}
```

```
writer.close();
FileReader reader = new FileReader(file);
int character;
while ((character = reader.read()) != -1) {
    System.out.print((char) character);
}
reader.close();
} catch (IOException e) {
    System.out.println("An error occurred.");
    e.printStackTrace();
}
}
}
```

Best Practices Implementing Input/Output Operations

Here are some best practices to keep in mind when implementing input/output operations in Java:

Always close resources: Always close the input/output streams, file readers, and writers after you're done using them. This helps in preventing resource leaks and making sure that the program runs efficiently.

Use buffered streams: Buffered streams are used to improve the performance of input/output operations in Java. Always use buffered streams instead of unbuffered streams to speed up I/O operations.

Use descriptive variable names: Use descriptive variable names when working with input/output operations in Java. This makes the code easier to read and understand, reducing the likelihood of errors and bugs.

Handle exceptions: Always handle exceptions when working with input/output operations in Java. This helps in making sure that the program runs smoothly and that the user is informed of any errors that occur.

Conclusion

In summary, files and input/output operations are essential features in Java that enable developers to store data permanently and read and write data from various sources. By following best practices for implementing input/output operations, developers can ensure that their Java programs are efficient, maintainable, and error-free.