

**Lab Manual- Java Streams for reading from and writing to files with Exception Handling**

**Prepared for**:

**Date:** 18th June 2024

**Prepared by:**

Document Name: Lab Manual **Document Number** SDLab333

**Contributor:**

Contents

[1. Streaming in Java 3](#_Toc172439880)

[1. InputStream class 4](#_Toc172439881)

[2. OutputStream class 4](#_Toc172439882)

[2. Create Sample Program to Understand Stream 5](#_Toc172439883)

[2.1 Reading from a File 5](#_Toc172439884)

[2.2 Writing to a File 5](#_Toc172439885)

[2.3 Explanation: 6](#_Toc172439886)

[2.4 Create the Input File 6](#_Toc172439887)

[2.5 Compile and Run the Java Code for Reading 7](#_Toc172439888)

[2.6 Create the Output File (Optional) 7](#_Toc172439889)

[3. Create Book.java and ExampleBooks.java to Read BookName and Author from Input File 8](#_Toc172439890)

[3.1 Create Book.java 8](#_Toc172439891)

[3.2 Create ExampleBooks.java 9](#_Toc172439892)

[3.3 Create the Input File books.txt 9](#_Toc172439893)

[3.4 Compile and Run 10](#_Toc172439894)

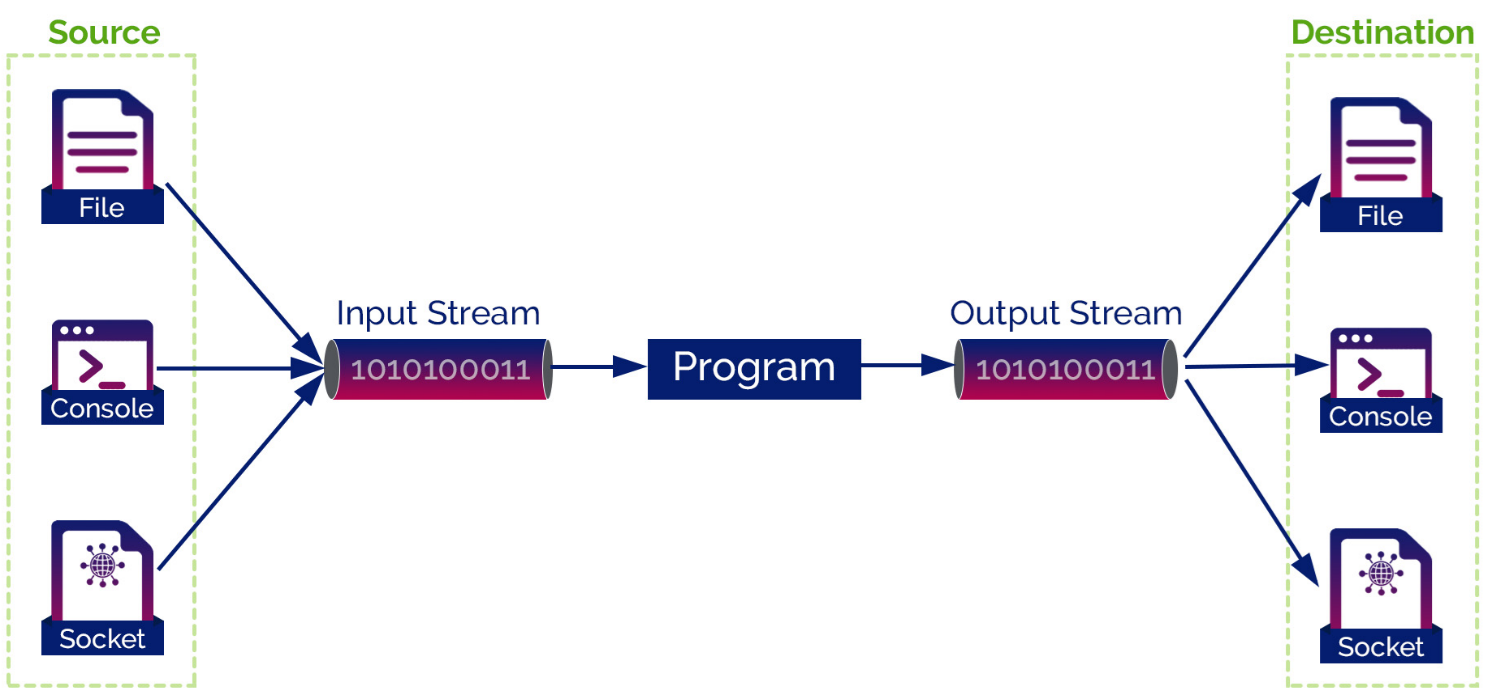
# Streaming in Java

In java, the **IO operations** are performed using the **concept of streams**. Generally, **a stream means** a **continuous flow of data**.

In java, a stream is a logical container of data that allows us to read from and write to it. A stream can be linked to a **data source, or data destination, like a console, file or network connection** by java IO system. The stream-based IO operations are faster than normal IO operations.

The **Stream** is defined in the **java.io package**.

To understand the functionality of java streams, look at the following picture.



In java, the stream-based IO operations are performed using two separate streams input stream and output stream.

* The **input stream** is used for input operations,
* The **output stream** is used for output operations.

The java stream is composed of bytes.

In Java, every program creates 3 streams automatically, and these streams are attached to the console.

* **System.out**: standard output stream for console output operations.
* **System.in**: standard input stream for console input operations.
* **System.err**: standard error stream for console error output operations.

## InputStream class

The InputStream class has defined as an abstract class, and it has the following methods which have implemented by its concrete classes.

| **S.No.** | **Method with Description** |
| --- | --- |
| 1 | **int available()**  It returns the number of bytes that can be read from the input stream. |
| 2 | **int read()**  It reads the next byte from the input stream. |
| 3 | **int read(byte[] b)**  It reads a chunk of bytes from the input stream and store them in its byte array, b. |
| 4 | **void close()**  It closes the input stream and also frees any resources connected with this input stream. |

## OutputStream class

The OutputStream class has defined as an abstract class, and it has the following methods which have implemented by its concrete classes.

| **S.No.** | **Method with Description** |
| --- | --- |
| 1 | **void write(int n)**  It writes byte(contained in an int) to the output stream. |
| 2 | **void write(byte[] b)**  It writes a whole byte array(b) to the output stream. |
| 3 | **void flush()**  It flushes the output steam by forcing out buffered bytes to be written out. |
| 4 | **void close()**  It closes the output stream and also frees any resources connected with this output stream. |

# Create Sample Program to Understand Stream

Here's a simple example of using **Java Streams** for reading from and writing to files. This example demonstrates how to read from a file, process its content, and write the results to another file using Java Streams.

### Reading from a File

First, let's create a method to read from a file and process its content:

**FileReadExample.java**

import java.nio.file.Files;

import java.nio.file.Paths;

import java.util.List;

import java.util.stream.Collectors;

public class FileReadExample {

    public static void main(String[] args) {

        String inputFilePath = "input.txt"; // Path to your input file

        try {

            // Read all lines from the file into a List of Strings

            List<String> lines = Files.lines(Paths.get(inputFilePath))

                                      .collect(Collectors.toList());

            // Print each line to the console

            lines.forEach(System.out::println);

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

}

### Writing to a File

Now, let's create a method to write some content to a file:

**FileWriteExample.java**

import java.nio.file.Files;

import java.nio.file.Paths;

import java.util.List;

public class FileWriteExample {

    public static void main(String[] args) {

        String outputFilePath = "output.txt"; // Path to your output file

        List<String> linesToWrite = List.of("Hello, world!", "This is a test.");

        try {

            // Write the list of lines to the file

            Files.write(Paths.get(outputFilePath), linesToWrite);

            System.out.println("Data written to file successfully.");

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

}

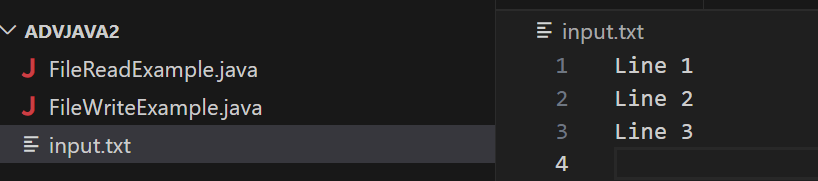
### Explanation:

1. **Reading from a File**:
   * **Files.lines(Paths.get(inputFilePath))** reads lines from the specified file path into a Stream<String>.
   * **collect(Collectors.toList())** collects the lines into a List<String>.
   * **lines.forEach(System.out::println)** prints each line.
2. **Writing to a File**:
   * **Files.write(Paths.get(outputFilePath),** **linesToWrite)** writes the list of strings to the specified file path.

### Create the Input File

You need to ensure that the files exist and are correctly placed for reading and writing. Here’s a step-by-step guide to creating and using these files:

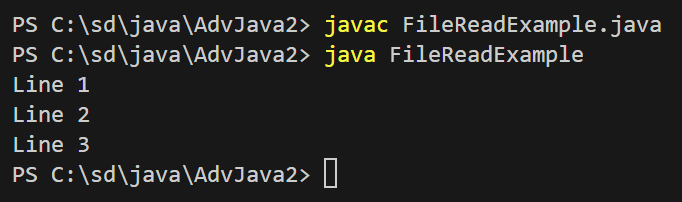
* Create a new text file named **input.txt.**
* Open **input.txt** in a text editor (such as Notepad on Windows, TextEdit on macOS, or any code editor).
* Add some text to the file, such as:



### Compile and Run the Java Code for Reading

**javac FileReadExample.java**

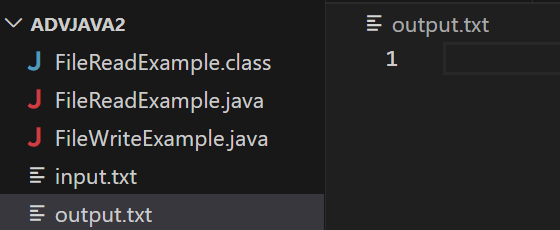
**java FileReadExample**

****

### Create the Output File (Optional)

If you want to create the output file manually before running the writing code, you can do so:

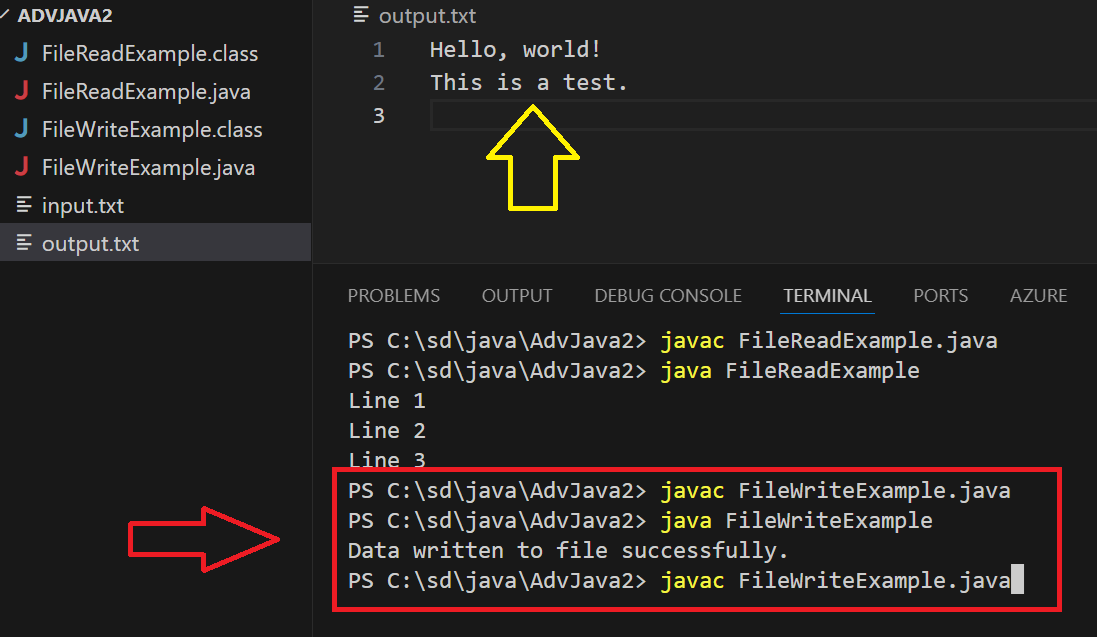
* Create a new text file named **output.txt**.
* You can leave the file empty or add some initial content

****

**javac FileWriteExample.java**

**java FileWriteExample**

The program should write the content to output.txt.

****

# Create Book.java and ExampleBooks.java to Read BookName and Author from Input File

### Create Book.java

This class will represent a book with a name and an author.

// Book.java

public class Book {

    private String name;

    private String author;

    public Book(String name, String author) {

        this.name = name;

        this.author = author;

    }

    public String getName() {

        return name;

    }

    public String getAuthor() {

        return author;

    }

    @Override

    public String toString() {

        return "Book{name='" + name + "', author='" + author + "'}";

    }

}

### Create ExampleBooks.java

This class will read book information from a text file, create Book objects, and print them out.

// ExampleBooks.java

import java.nio.file.Files;

import java.nio.file.Paths;

import java.util.List;

import java.util.stream.Collectors;

public class ExampleBooks {

    public static void main(String[] args) {

        String inputFilePath = "books.txt"; // Path to your input file

        try {

            // Read lines from the file

            List<String> lines = Files.lines(Paths.get(inputFilePath))

                                      .collect(Collectors.toList());

            // Process each line

            for (String line : lines) {

                // Assume each line is in the format "BookName,AuthorName"

                String[] parts = line.split(",");

                if (parts.length == 2) {

                    String name = parts[0].trim();

                    String author = parts[1].trim();

                    Book book = new Book(name, author);

                    System.out.println(book);

                } else {

                    System.out.println("Invalid line format: " + line);

                }

            }

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

}

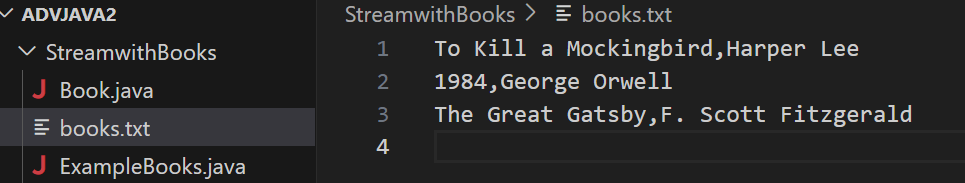
### Create the Input File books.txt

Create a text file named **books.txt** in the same directory as your Java files. Add book information in the following format:

To Kill a Mockingbird,Harper Lee

1984,George Orwell

The Great Gatsby,F. Scott Fitzgerald



### Compile and Run

Save the Book.java and ExampleBooks.java files in the same directory and Execute Below command

javac Book.java ExampleBooks.java

java ExampleBooks

