1DV532 – Starting Out with Java

Lesson 11

File Input and Output

Dr. Nadeem Abbas
nadeem.abbas@lnu.se

Linnæus University

Streams

- A *stream* is simply a flow of data between a program and some Input/Output (I/O) device or file.

- There are two fundamental types of Streams
  1. Input Stream
  2. Output Stream

Input Stream

- If the data flow from a data source, such as keyboard or a text file, into a program, then the stream is called an *input stream*
  - As shown in the figure below,the input stream is used to **read data** from a data source to a program.
  - `System.in` is a standard input stream provided by Java.
    - This stream is already open and ready to supply input data.
    - Typically this stream corresponds to keyboard input

```
Scanner keyboard = new Scanner(System.in);
```

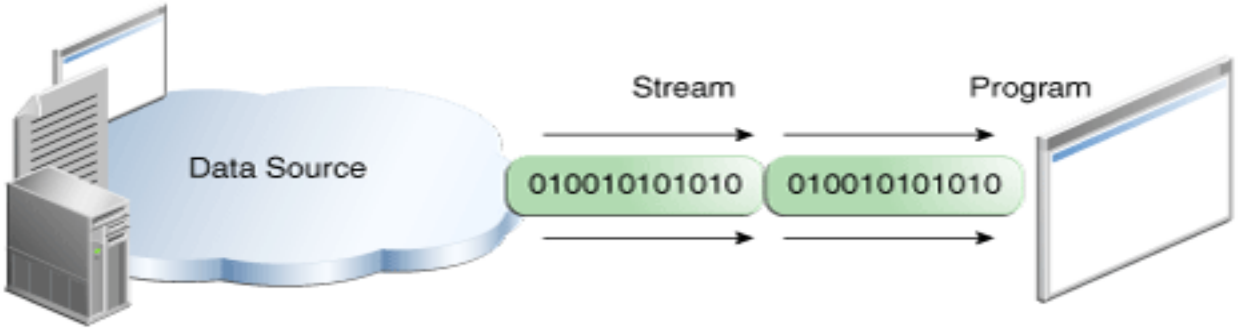

*Image taken from https://docs.oracle.com/javase/tutorial/essential/io/streams.html

Output Stream

- If the data flow out of a program to an output device or a text file, then the stream is called an *output stream*
  - As shown in the figure below,the output streams is used to **write data** from a program to a data source or an output device.
  - `System.out` is a standard out stream provided by Java.
    - This stream is already open and ready to accept output data.
    - Typically this stream corresponds to display output screen.

```
System.out.println(data);
```

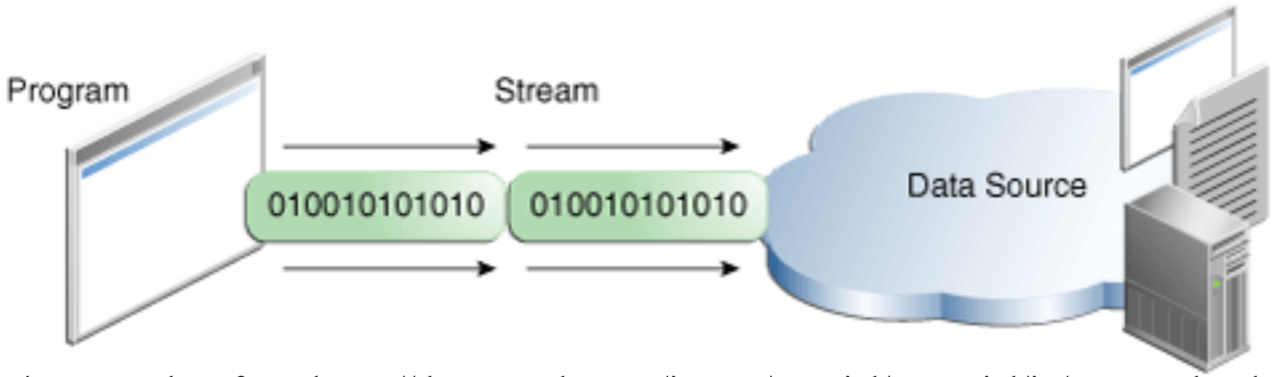

*Image taken from https://docs.oracle.com/javase/tutorial/essential/io/streams.html

File Input and Output (I/O)

- In Lesson 5, step 1, we learned Console I/O
  - how to input or read data from a keyboard into a java program, and
  - how to output or write data from a java program to a console or a display device.

- In this lesson, we learn File I/O
  - how to input or read data from a **File** into a java program, and
  - how to output or write data from a java program to a **File**

File Input and Output (I/O)

- Java supports File I/O for two basic types of files
  1. Text Files
  2. Binary Files

- **Text Files** are designed to be read by human beings, and can be read or written using a text editor are called *text files*
  - Plain text files with extension .*txt* are often used for File I/O in java

- **Binary Files** are designed to be read by programs thus consist of a sequence of binary digits.
  - .*bin* and .*dat* are the commonly used extensions for binary files

- For the File I/O, in this course, we focus only on *Text Files.*

Writing to a Text File

- **PrintWriter,** a Java library class found in package `java.io`, is often used to write to a text file
  - The **`PrintWriter`** class has **`print`** and **`println`** methods that work similar to the **`System.out. print`** and **`System.out. println`** methods to output or write data to a text file

- An object of *PrintWriter* type is created and connected to a text file for writing as follows:

```
FileOutputStream fileOS = new FileOutputStream (”sample.txt")
PrintWriter writer = new PrintWriter(fileOS );
```

Writing to a Text File

- Constructor of the PrintWriter class requires an object of **`FileOutputStream`** to be passed an input parameter:

```
PrintWriter writer = new PrintWriter(fileOS);
```

  – `fileOS` in the above statement is an object of type *FileOutputStream*

- **`FileOutputStream`** is an output stream for writing data to a File.
  – An object of type FileOutputStream is created by passing a string representing the **file name** as its argument:

```
FileOutputStream fileOS = new FileOutputStream("sample.txt");
```

  – **`sample.txt`** is the name of the file to which we want to write

- Example program, *TextFileOutputDemo.java*, demostrates use of the PrintWriter class to write data to a text file

Example Program – TextFileOutputDemo.java

```
import java.io.PrintWriter;
import java.io.FileOutputStream;
import java.io.FileNotFoundException;
public class TextFileOutputDemo {
    public static void main(String[] args) {
          PrintWriter writer = null;
          FileOutputStream fileOS;
          try {
                  fileOS = new FileOutputStream("sample.txt");
                  writer = new PrintWriter(fileOS);
          } catch (FileNotFoundException e) {
                  System.out.println("Error opening the file sample.txt.");
                  System.exit(0);
          }
          System.out.println("Writing to file.");
          writer.println("This is justs and example file created to
    demonstrate how we can write data to files from a java program.");
          writer.println("Any text can be written here...");

          writer.close();
          System.out.println("End of program.");
    }
}
```

Writing to a Text File

```
FileOutputStream fileOS = new FileOutputStream (”sample.txt")
PrintWriter writer = new PrintWriter(fileOS );
```

- The above two statements results in an object of the class `PrintWriter` that is connected to the file *`“sample.txt”`*
  - The process of connecting a stream to a file is called ***opening the file***
  - If the file already exists, then doing this causes the old contents to be lost
  - If the file does not exist, then a new, empty file named with given name, e.g., “*sample.txt*” is created
- After creating `PrintWriter` object, we inovke its `print` and `println` methods for writing data to the file

FileNotFoundException

- While opening a File i.e., connecting file to output stream, a **`FileNotFoundException`** can be thrown
  - In this context it actually means that the file could not be created
  - This type of exception can also be thrown when a program attempts to open a file for reading and there is no such file
- It is therefore necessary to enclose the code that opens a file in Exception Handling blocks
  - The file should be opened inside a **`try`** block
  - A **`catch`** block should catch and handle the possible exception
- **`FileNotFoundException`** is a type of Exception, **`try`** and **`catch`** are exception handling statements.
- We will learn about Exceptions and Exception Handling in next step
  - For now, you should assume that these are Java constructs used to handle abnormal events that occur at runtime, for example, file not found.

Reading From a Text File

- The class `Scanner` can be used for reading from the keyboard as well as reading from a Text File
  - To read from a Text File, we need to replace the argument `System.in` (to the `Scanner` constructor) with a suitable stream that is connected to the text file that we want to read

```
Scanner inputStream=
   new Scanner(new FileInputStream(FileName));
```

- Methods of the `Scanner` class for reading input behave the same whether reading from the keyboard or reading from a text file
  - For example, the `nextInt` and `nextLine` methods
- Example program, *TextFileInputDemo.java*, demonstrates how may we use the Scanner class to read data from a file.

Example Program – TextFileInputDemo.java

```
import java.util.Scanner;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
public class TextFileInputDemo {
   public static void main(String[] args) {
         String fileName="sample.txt"; //name of the file for reading
         Scanner inputStream = null;
         try {
                  inputStream = new Scanner(new FileInputStream(fileName));
         } catch (FileNotFoundException e) {
                  System.out.println("File "+fileName+" was not found");
                  System.out.println("or could not be opened.");
                  System.exit(0);
         }
         System.out.println("Lines read from a File "+fileName+ " are:");
         while (inputStream.hasNext()) { // true ==> more text available
                  String str = inputStream.nextLine(); // read next line
                  System.out.println(str);
         }
         inputStream.close();
   }
}
```

Reading From a Text File

- An object of *Scanner* type is created and connected to a text file as follows:

```
Scanner inputStream =
 new Scanner(new FileInputStream(fileName));
```

- As shown above, constructor of the Scanner class requires an object of **`FileInputStream`** to be passed an input parameter
- **`FileInputStream`** is an input stream for reading data from a File.
  - An object of type FileInputStream is created by passing a string representing the file name as its argument as follows:

```
FileInputStream fileIS = new FileInputStream("sample.txt");
```

Reading From a Text File

- The try and catch block are placed in the example code to handle `FileNotFoundException` which can be thrown if the file we are attempting to read does not exist.

Close the Stream

- When a program has finished writing to a file, or reading from a file, it should always close the stream connected to that file

```
writer.close();
inputStream.close();
```

  - This allows the system to release any resources used to connect the stream to the file
  - If the program does not close the file before the program ends, Java will close it automatically, but it is safest to close it explicitly

Suggested Readings

- Absolute Java, Global Edition, 6/E by Walter J. Savitch, Chap 10 File I/O

- Java Tutorials
  - https://docs.oracle.com/javase/tutorial/essential/io/index.html

Lnu.se