File I/O

Contributor: **Arshia, Lucas**

Main contents:

- **How do you read/write from a file?**
    - Import
    - Try-catch structure
    - Create the objects
    - Difference between bufferedReader and FileReader (same with Writer)
    - Methods
    - inputStreamReader and inputStreamWriter
    - Create the file using Java code
- **When would reading / writing be important?**
- **What are good practices when opening and closing the file?**
- **Demo code**
- **Questions:**
    - Do we need to close the file after writing/reading from it?
    - There is a CSV file, could you try to read the file and figure out the highest high and lowest low on the specific date? And print it out.

How do you read/write from a file?

There are two simple ways to read and write from the file:

FileReader/BufferedReader and FileWriter and BufferedWriter (first way):

https://docs.oracle.com/javase/7/docs/api/java/io/FileReader.html

https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html

The necessary import:

import java.io.FileWriter;

import java.io.IOException;

import java.io.FileReader;

import java.io.BufferedReader;

import java.io.BufferedWriter;

Other import:

import java.io.FileNotFoundException;

Try-catch: (using it in file input and output both)

When we use file input and output, we usually use the try-catch.

Remember the "IOException e" should be include in the bracket

Format:

Try{
//your code
}
Catch (IOException e){
System.out.println("error,cannot read the file");
}

```
try{
  //your code
}
catch(IOException e){
    System.out.println("error, cannot read the file");
}
```

After preparing, there will be how to use file input and output below:

For creating object of FileReader / BufferedReader/ FileWriter / BufferedReader:

The format of creating should be:

// Creates a FileReader

FileReader fileRead = new FileReader(file);

// Creates a BufferedReader

BufferedReader input = new BufferedReader(fileRead);

FileWriter fileWriter = new FileWriter (file);

BufferedWriter output = new BufferedWriter(fileWriter);

```
// Creates a FileReader
FileReader fileRead = new FileReader(file);
// Creates a BufferedReader
BufferedReader input = new BufferedReader(fileRead);
```

```
FileWriter fileWriter = new FileWriter(file);
BufferedWriter output = new BufferedWriter(fileWriter);
```

The FileReader is the type and fileRead is the object, users could define them as any words they want. The “file” word in the brackets means the location of your file or you could just write the name of the file you want to read such as “file.txt”. In addition, a user could create a file in a java program, and the system will create a new file in the local folder same with default location.

For the BufferedReader, create the object with the proper name you want and the fileRead in the brackets means the fileReader object, if you use another object name, the user has to change it as the object name you create.

FileWriter and BufferedWriter are the same as Reader.

Difference between BufferedReader and why use BufferedReader:

During the write operation, the characters are written to the internal buffer instead of the disk. Once the buffer is filled or the writer is closed, the whole characters in the buffer are written to the disk.

Buffer stores data temporarily, it has many advantages than fileReader:

Basis	BufferedReader	FileReader
Use	It is used to read characters from any type of character input stream (String, Files, etc.)	It can be used only for reading files
Buffer	Uses Buffer internally to read characters from	Doesn't use Buffer. Directly reads from the file by accessing the hard drive.
Speed	Faster	Slower
Efficiency	Much more efficient for reading files	Less efficient
Reading Lines	BufferedReader can be used to read a single character at a time as well as a line at a time.	It can read only one character at a time, can not read lines

Reference: https://www.geeksforgeeks.org/difference-between-bufferedreader-and-filereader-in-java/

Methods:

obj.read()	Read single character
obj.readLine()	Read whole line
obj.write()	Write a single character
obj.writeLine()	Write a line separator
obj.newLine()	Write a new line

obj.close()	Close the FileReader / BufferedReader / FileWriter / BufferedWriter

Use those methods to achieve programmer's goals.

Do not forget to close all FileReader or Writer at the end of the "try" part.

InputStreamReader / OutputStream: (Second way)

https://docs.oracle.com/javase/7/docs/api/java/io/InputStream.html

We do not learn from grade 11, but I mention this method in content of learning.

Very similar with BufferedReader and FileReader

Use the try-catch and create the objects in the same way:

```
Try{
     FileInputStream obj_defined = new FileInputStream("your file name");
     InputStreamReader in = new InputStreamReader(obj_defined);
}
Catch (IOException e){
      System.out.println("error, cannot read the file");
}
```

```
Try{
    FileOutputStream file = new FileOutputStream(String path);
    OutputStreamWriter output = new OutputStreamWriter(file);
}
Catch (IOException e){
    System.out.println("error, cannot read the file");
}
```

```
try{
   FileInputStream obj_defined = new FileInputStream("your file name");
   InputStreamReader in = new InputStreamReader(obj_defined);
}
catch(IOException e){
    System.out.println("error, cannot read the file");
}
```

```
try{
   FileOutputStream file = new FileOutputStream(String path);
   OutputStreamWriter output = new OutputStreamWriter(file);
}
catch(IOException e){
    System.out.println("error, cannot read the file");
}
```

Method:

Obj.read()	Read a single character
obj.ready()	Tells whether this steam is ready to be read
obj.flush()	Flushes the stream
obj.write()	Write a single character to the file

Do not forget to close them!

Difference:

BufferedReader reads a couple of characters from the Input Stream and stores them in a buffer.

InputStreamReader reads only one character from the input stream and the remaining characters still remain in the streams hence There is no buffer in this case.

Addition:

There is a way to create a file using Java:

- Import java.io.File;
- Create the object with File type, and write the file name in the brackets

Import java.io.File;

Import java.io.IOException;

```
Public class DemoReading {
    Public static void main(String[] args){
          Try{
               File file = new File(“fileName.txt”);
          }
          Catch (IOException e){
               System.out.println(“error, cannot create file”);
          }
    }
}
```

```java
import java.io.File;
import java.io.IOException;

public class DemoReading {
    public static void main(String[] args) {
        try {
            //create a file with the name in local default destination
            File file = new File("fileName.txt");
        }
        catch(IOException e) {
            System.out.println("error, cannot create file");
        }
    }
}
```

When would reading/writing to a file be important?

One important occasion is when programmers need to read values from other files with different types such as the CSV file we learned in grade 11. If a programmer wants to use the greatest value in the file, they could use file read and split to store the highest value to the corresponding user-defined variables to use later instead of the programmer finding the highest one by one. The reason file reading is important is that the programmer could use arithmetic and loops to read the value they want with the help of the system. I believe that will be important especially with a real-life example, assuming that you are a programmer in a company, your teammates who write a data form give that to you and file input and output could help you to

achieve your goals easily. That goal could be diversity based on how you achieve that. File write is also important, it could help you to write the data in a specific location in different types of the file rather than the programmer opening the file and finding the specific destination to input the data.

What are good practices when opening and closing a file?

There are reminders and good practices when you start your file I/O:

- Import corresponding classes and packages.
    - java.io.FileWriter
    - java.IOException
    - java.io.FileReader
    - java.io.BufferedReader
    - java.io.BufferedWriter
- Close the Reader and Writer since you are done your code with the object you create (Both File and Buffer)
    - obj.close();
- Notice the file location and file name when creating object or files
- Make sure if you use bufferedReader, the fileReader object should be in the brackets
- Leave a message or other operation in Catch blocks to help you know where is the error
- The system could only write string to the file, please remember to cast the value if you are trying to write a number into the file
- Same as write, system will read the file as string type, please remember cast it if you want to use the number inside the file

    - Integer.parseInt(String); //or other cast methods

- Check the file if it is follow your expectation
- Always remember your try-catch structure! Or there will be an error

Demo Code:

FileWrite:

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class DemoWriting {
  public static void main(String[] args) {
     String data = "This is the data in the output file";
    int num = 42;

    try {
     //will try to write to output.txt. If it doesn't exist, it will create it
     File file = new File("output.txt");
     FileWriter fileWriter = new FileWriter(file);
     BufferedWriter output = new BufferedWriter(fileWriter);

     output.write(data);
     output.newLine();
     output.write("" + num);

     output.close();
     fileWriter.close();
   }
   catch (IOException e) {
     System.out.println("Sorry, cannot write to that file.");
   }
  }
}
```

```java
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class DemoWriting {
  public static void main(String[] args) {

    String data = "This is the data in the output file";
    int num = 42;

    try {
      File file = new File("output.txt");
      FileWriter fileWriter = new FileWriter(file);
      BufferedWriter output = new BufferedWriter(fileWriter);

      output.write(data);
      output.newLine();
      output.write("" + num); //cast the number to string so that it could be writen in the file

      output.close();
      fileWriter.close();
    }
    catch (IOException e) {
      System.out.println("Sorry, cannot write to that file.");
    }
  }
}
```

output.newLine();

Create a new Line so that system could write next data in next Line

output.write(“” + num);

This is a way to cast numbers to strings.

The “output.txt” file looks like:

```
1 This is the data in the output file
2 42
3
```

FileRead:

The “input.txt” file looks like:

```
1 13
2 15
3 28
4 7
5 8
6 909
```

The Demo of file Reading:

```
import java.io.FileReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
import java.io.BufferedReader;

public class DemoReading {
  public static void main(String[] args) {
    // Creates an string to read a line
    String line;
    try {
      //will try to write to output.txt. If it doesn't exist, it will create it
      File file = new File("input.txt");
      // Creates a FileReader
      FileReader fileRead = new FileReader(file);
      // Creates a BufferedReader
      BufferedReader input = new BufferedReader(fileRead);

      // Reads lines of the text using a loop, while there is still text
      while ((line = input.readLine()) != null) {

        //prints out the line read
        //all input is String type
        System.out.println(line);
      }
      // Closes the reader
      input.close();
      fileRead.close();
    }

    catch(IOException e) {
      System.out.println("Cannot read from file.");
    }
```

```
  }
}
```

```
import java.io.FileReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.BufferedReader;

public class DemoReading {
  public static void main(String[] args) {

    String line;

    try {
      File file = new File("input.txt");
      FileReader fileRead = new FileReader(file);
      BufferedReader input = new BufferedReader(fileRead);

      while ((line = input.readLine()) != null) {
        System.out.println(line);
      }

      input.close();
      fileRead.close();
    }

    catch(IOException e) {
      System.out.println("Cannot read from file.");
    }
  }
}
```

If there is still a value in the next line, the system will continue to print each line's information which is controlled by a while loop.

After running, the console looks like:

```
  ----jGRASP exec: java DemoReading
13
15
28
7
8
909

  ----jGRASP: operation complete.
```

Questions

1. Do we need to close the file after writing/reading from it?
2. There is a CSV file, could you try to read the file and figure out the highest high and lowest low on the specific date? And print it out.

CSV file:

https://drive.google.com/file/d/15O3Umx1ikfYXsP2pTrSfAjZlaxSYsz5w/view

Answers

Citation

https://www.geeksforgeeks.org/file-handling-java-using-filewriter-filereader/
https://www.geeksforgeeks.org/scanner-class-in-java/
https://docs.oracle.com/javase/7/docs/api/java/io/InputStream.html\
https://docs.oracle.com/javase/7/docs/api/java/io/FileReader.html
https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html
https://www.java67.com/2020/02/40-java-io-and-files-interview-questions-answers.html
http://www.iitk.ac.in/esc101/05Aug/tutorial/essential/io/QandE/answers.html