Samuel Hossain
CSCI 260- M05/6
Professor. Fischman
CSCI Team Project

Reading Text Files

Objective:

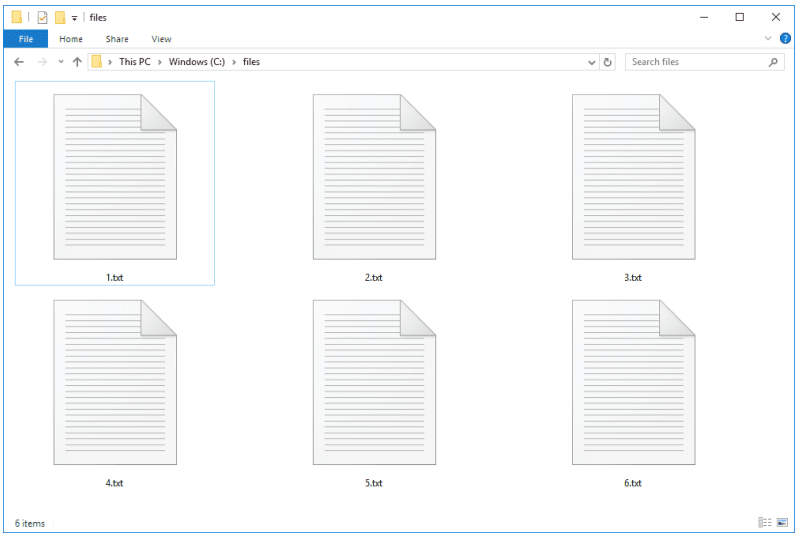
In this section, you should be able to use four ways to read a plain text file in Java:

- Read text files using the **BufferedReader** Class.
- Read text files using the **FilesReader** Class.
- Read text files using **Scanner.**

Description of Text Files:

Most of your data in any file is stored in binary digits, known as **Bits**. However, that is not the case in this topic. We think of them as a sequence of characters. We called them **Text Files**. It is a computer file that is structured as a sequence of lines in an electronic text. It is like writing on a loose-leaf paper but in an electronic version.

Most of your java programs are stored in a text file. Those files appear the same on all computers. You could use the same file in any computer with less errors. **Reading a text file** is like **reading a binary file**. It depends on which classes we are using to perform the input and output methods. The benefit of using text files is that you can create, look at, and edit them by using a text editor. It includes a small size and can be opened on different platforms such as **Windows 10, Mac, and Linux**. They are easier and faster to create. One thing Java is useful is because it provides several mechanisms to read from files. Each class has its own package and method to make sure that the code can run properly. Below is an example of a text file that looks like in a document form. It is always in .txt format.



Three ways of reading text files in Java:

You can use the **FileReader**, **BufferedReader**, or **Scanner** to read a text file.

Using the BufferedReader Class (reading text files):

We can use the **BufferedReader** class to read text files from an Input Stream. For those who do not know this class, it reads text from a character-input stream, buffering characters to provide the efficient reading of characters, arrays, and lines. It is common if you are working on multiple small files. Each read request made of a Reader causes a corresponding request to be made of the underlying byte stream. For BufferedReader's **ReadLine(),** it reads a line of text.

How to read file in Java (using BufferedReader Method):

We used the **BufferedReader** class, it minimizes the number of operations by reading parts of characters and stores them in a buffer. Here is the example of the code:

```
package teamproject;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class main {

    public static void main(String[] args) throws Exception
    {
        File file = new File("C:\\Users\\Samuel Hossain\\Desktop\\test.txt");

        BufferedReader br = new BufferedReader(new FileReader(file));

        boolean st;
        while ((st = br.readLine() != null))
            System.out.println(st);
    }
}
```

Using the FileReader class (reading text files):

We will be using this method if you want to read streams of characters. It comes with a **java.io** package. Once we have done that, it **extends** the **InputStreamReader** class and **implements Readable interface**. For those who do not know, **FileReader** is used to read data from files.

How to read file in Java (using FileReader's Method):

To use **FileReader**, we import the package **java.io** using the **import** statement. We use the constructors to create the instance of **FileReader**. There are two examples to read a file using **FileReader.** This is convenient for reading character files by using one of these constructors.

```
import java.io.FileReader;
import java.io.IOException;

public class main {

    public static void main(String [] args)  {

        File file = new File("test.txt");

        try (FileReader fr = new FileReader(file))
        {
            int content;
            while ((content = fr.read()) != -1) {
                System.out.println((char) content);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }



    }
```

Reading line by line example:

```
1  package teamproject;
2  import java.io.FileReader;
3  import java.io.IOException;
4
5
6  public class main
7  {
8      public static void main(String[] args) throws Exception
9      {
0          String fileName = "test.txt";
1
2          FileReader fileReader = new FileReader(fileName);
3
4              int i;
5              while((i = fileReader.read()) != -1) {
6                 System.out.print((char)i);
7              }
8          }
9      }
0
1
```

Using the Scanner method (reading text files):

We will use the **Scanner** method to read text files. For those who do not know a Scanner, it is a class that is used for obtaining the input of the primitive types like int, double, etc. It breaks its input into tokens using this type of pattern. It may be converted into values of different types using the various methods. In this tutorial, we will determine how we can use the **Scanner** method to read files in Java. It is a utility class in **java.util** package and gives methods to read int, long, String, double, etc**.** from other sources. It is an easy way to read any user input using **System.in** as a source.

How to read file in Java (using Scanner Method):

We have created a **File** to represent a text file in Java. Once we have done that, we pass the file onto the **Java.util.Scanner** for scanning the file itself. When using the **Scanner,** it provides methods such as **hasNextLine()** and **readNextLine()**. Those two methods can be used to read line by line. Be sure to check your line-by-line code before you execute it. Here's the example of the code:

```java
package teamproject;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.nio.file.Files;
import java.util.Scanner;


public class main {

    public static void main(String[] args) throws Exception {

        System.out.println("Reading a text line by line: ");
        Scanner sc = new Scanner(new File("file.txt"));
        while(sc.hasNext()) {
            String str = sc.nextLine();
            System.out.println(str);
        }
    }
}
```

Entire Code:

```java
package teamproject;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.nio.file.Files;
import java.util.Scanner;


public class main {

    public static void main(String[] args) throws Exception {

        System.out.println("Reading the whole text: ");
        Scanner sc = new Scanner(new File("file.txt"));
        while(sc.hasNext()) {
            String word = sc.nextLine();
            System.out.println(word);
        }

        sc.close();
    }
}
```

Jhoselyn Moro-Romero
CSCI 260- M05/6
Professor. Fischman
CSCI Team Project

Writing Text Files

Objective:

This section of the book is the continuation and complement of the **Reading Text Files** chapter. In this chapter we will be writing text files using the following:

- Write a text file using the **FileWriter** class.
- Write a text file using the **BufferedWriter** class.
- Write a text file using the **Scanner** class.

Description of Text Files:

The majority of data stored within any file is stored as **Bits**. **Bits** are the most basic way information is stored, they only have two possible values; 0 or 1. **Text files**, whether it be writing or reading, can be written using letters. As people we understand things are strings, letters and characters in succession of each other and text files allow us to write this way within the stored files.

Writing text files on your computer is almost the same as writing on a piece of paper, you just need to be mindful of the classes, input and output methods you will be using to get the program to make your file for you. In some cases if you know what classes to use you can ask for a user input allowing you to write whatever you want without you having to write your original note within the program itself. To do this you have to implement the **Scanner** class. Writing or reading text files on your computer isn't exclusive to programming, if you need to you can always access the 'Notepad' on any machine with a **Windows**, **Mac** or **Linux** operating system (**OS**).

Something to note with Text Files would be that when you save them to view some other time the files itself will always be in a .txt format. When you try to look for it after make sure you either access the notepad directly or look for the name of the file with .txt after it.

Ways to Write a File:

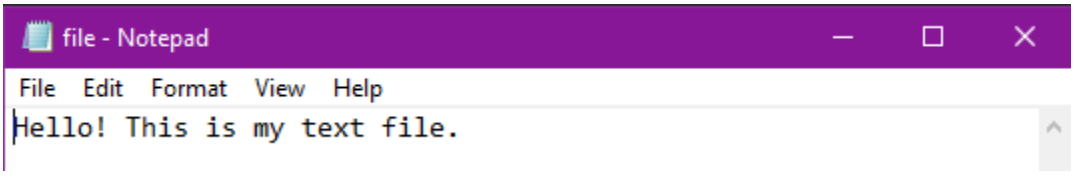
The classes that we will be using to write the Text files are **FileWriter**, **BufferedWriter** and **Scanner**.

Using the FileWriter Class:

We can use this class when we want to write our notes into the program itself. When you are programming you will be able to find this method in the **java.io** package from here you can then implement it into your program.

To implement the **FileWriter** class into the program you need to import it, do this by simply typing **import** followed by **java.io.FileWriter**. Through the **FileWriter** method you can directly write the data into the file.

```
package teamproject;
import java.io.FileWriter;



public class Main {

    public static void main(String args[]) {

        String data = "Hello! This is my text file.";

        try {

            FileWriter output = new FileWriter("file.txt");

            output.write(data);

            output.close();

        }

        catch (Exception e) {

            e.getStackTrace();

        }

    }

}
```



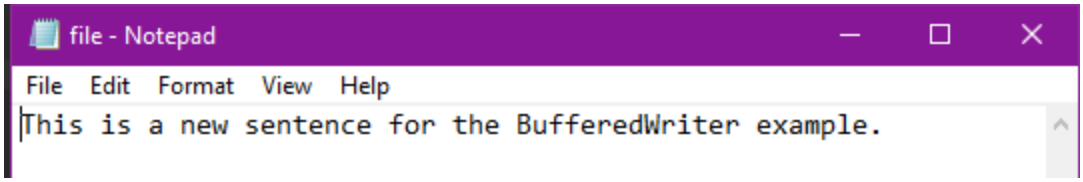
Using the BufferedWriter class:

Using the **BufferedWriter** allows us to do almost the exact same thing that the **FileWriter** method does except more efficiently. You can find this class in the **java.io** package. To implement the **BufferedWriter** into your program you do the same thing as we did when we were implementing the **FileWriter** class. Using the **BufferedWriter** class allows us to write data as letters more efficiently.

The best way I can explain the buffer would be that instead of the data being stored inside the disk it is stored within the buffer. This in turn retains all the data inside the buffer and then it is given to the disk, this in turn allows for less communication to the memory disk is reduced.

To explain the code below, the majority has not changed and something different would be that there is something there called a **flush**. Think of the **flush** as if you are getting rid of something, you are deleting something, this allows us to write something different in the text file we already have.

```
package teamproject;
import java.io.FileWriter;
import java.io.BufferedWriter;


public class Main {

    public static void main(String args[]) {

        String data = "This is a new sentence for the BufferedWriter example.";

        try {

            FileWriter file = new FileWriter("file.txt");

            BufferedWriter output = new BufferedWriter(file);

            output.write(data);

            output.flush();

            System.out.println("The data has been flushed.");

            output.close();

        }

        catch (Exception e) {

            e.getStackTrace();

        }
```



Using the Scanner Class:

The **Scanner** class allows us to ask for user input in the form of integers(**int**), double, strings, and etc.. For the purposes of this example we will just be writing any random thing we would like. The code below asks for the user input and then returns whatever you wrote preceded by the statement: "This is the data stored within the file: ", the **+name** followed by this statement tells the code to take the data you wrote and return it with the statement. The second image below shows what the output is for this code.

```
import java.util.Scanner;

public class Main {

    public static void main(String args[]) {

        Scanner input = new Scanner(System.in);

        System.out.println("Please enter what you would like to write: ");

        String name = input.nextLine();

        System.out.println("This is the data stored within the file: " + name);

        input.close();
```

```
Please enter what you would like to write:
Hello! This is the Scanner class example.
This is the data stored within the file: Hello! This is the Scanner class example.
```