Text File IO

Objectives

This short note explains how to perform basic text Input-Output (IO) using Java.

Introduction

- Our programs so far have been dealing with standard input (the Keyboard or *System.in*) and standard output (the Screen or *System.out*) for reading input and displaying output respectively.
- However, in real world applications, the data usually exist already in some file, so there is no need to type it in again using keyboard. Moreover, the output is generally needed not only for viewing but to be saved in a file for later review, further processing or printing.
- Thus, standard IO is generally not enough. Our programs must be able to read from a file and send output to a file.
- Accordingly, Java library comes with many classes for handling Input and Output for both Binary files (example images/videos, etc.) and Text files. These classes are mostly in the package java.io.
- In these short notes and examples, we explain only about Text File IO.

Writing to a Text File:

- Writing to a text file can be easily done using the PrintWriter class. The constructor of this class takes the name of the text file as argument. Example:

```
PrintWriter writer = new PrintWriter("Result.txt");
```

- The Constructor throws IOException, so the above object creation must be done inside a try-catch block.
- PrintWriter object has `print()` and `println()` methods. Therefore, it is used in exactly the same way as System.out object. The only difference is that the output goes to the text file associated with the object instead of the Screen.
- Another very important difference is that after you finish writing the data, then you must close the PrintWriter object using its `close()` method.

Example 1:

```
import java.io.*;
import java.util.Scanner;

public class WriteToFile {
    public static void main(String[] args) {
        int [] a = {1, 2, 3, 4, 5, 6, 7, 10};

        try {
            PrintWriter writer = new PrintWriter("Result.txt");
            for (int i=0; i<a.length; i++)
                writer.println(a[i]);
            writer.close();
        } catch (IOException ex) {
            System.out.println("File Error");
        }
    }
}
```

Reading from a Text File:

- Reading from a text file can be done using the same Scanner object that we have been using to read from keyboard. However, instead of passing *System.in* as argument to its constrictor, we must pass a File object which is associated with the text file we wish to read from.
- The constructor of the File class takes the name of the file as argument:

```
File file = new File("Result.txt");
Scanner reader = new Scanner(file);
```

Or combined the two lines above into one as:

```
Scanner reader = new Scanner(new File("Result.txt"));
```

- Once created, the scanner object behaves exactly as the one we are used to. It has methods, *nextInt(), nextDouble(), nextLine(), next(),* etc. for reading "input from the associated text file.
- One difference though is that, like PrintWriter object, the Scanner object associated with an input file must be closed using its *close()* method when it is done with.

Example 2:

```
import java.io.*;
import java.util.Scanner;

public class ReadFromFile {
    public static void main(String[] args) {
        try {
            Scanner reader = new Scanner(new File("Result.txt"));

            while (reader.hasNextInt())
                System.out.println(reader.nextInt());

            reader.close();
        } catch (IOException ex) {
            System.out.println("File Error");
        }
    }
}
```

Example 3:

```
import java.io.*;
import java.util.Scanner;

public class ReadTextFromFile {

    public static void main(String[] args) {
        try {
            Scanner reader = new Scanner(new File("AboutJUC.txt"));

            while (reader.hasNextLine())
                System.out.println(reader.nextLine());

            reader.close();
        } catch (IOException ex) {
            System.out.println("File Error");
        }
    }
}
```

Reading and Writing in one Program

- It is possible for a program to both read and write at the same time provided the input file and the output files are different. If the input/output file is the same, then then it must be closed after one operation before the other operation can be done.

Example 4:

```
import java.io.*;
import java.util.Scanner;

public class ReadAndWrite {
    public static void main(String[] args) {
        try {
            Scanner reader = new Scanner(new File("Result.txt"));
            PrintWriter writer = new PrintWriter("Result2.txt");

            while (reader.hasNextInt()) {
                int n = reader.nextInt();
                writer.println(n*n);
            }
            reader.close();
            writer.close();
        } catch (IOException ex) {
            System.out.println("File Error");
        }
    }
}
```

Overriding the Delimiters used by Scanner object

- By default, the Scanner object uses white spaces (any of ' ', \t, \n, \f, \r) as delimiters.
- This means if there is a non-white space character between the current int value and the next, then the nextInt() method cannot reach the next value.
- Similarly, the next() method will append punctuation characters such as ',', '.', etc. to the next word if the word is immediately followed by one of these.
- To avoid this problem, the useDelimiter() method is used to specify a different set of delimiters. The delimiters are specified as a regular expression in square bracket [..].
- Examples:
  - scanner.useDelimiter("[,\\s]"); //skip comer and white spaces.
  - scanner.useDelimiter("[\\p{Punct}\\s]+"); //skip punctuation characters and white spaces.

Example 4:

```
import java.io.*;
import java.util.Scanner;

public class ReadFromFile2 {
    public static void main(String[] args) {
        try {
            Scanner reader = new Scanner(new File("AboutJUC.txt"));

            reader.useDelimiter("[\\p{Punct}\\s]+");

            while (reader.hasNext())
                System.out.println(reader.next());

            reader.close();
        } catch (IOException ex) {
            System.out.println("File Error");
        }
    }
}
```