# ICS 102: Lab# 7 Text File I/O

**Objectives:**

Learn Java Text File I/O covering the following topics:

1. Introduction to Text-File I/O using Scanner, FileReader, and FileInputStream
2. Using loops in Text-file I/O

**File I/O**

**Introduction: Checked and Unchecked Exceptions**

Exceptions are errors that **may** be generated by the Java system when something goes wrong with a program. There are two types of exceptions:

* **Unchecked exceptions**: Are exceptions that are not checked for by the compiler.
* **Checked exceptions**: Are exceptions that are checked for by the compiler. A checked exception will give a compile error if such an exception may occur and is not handled or if the compiler is not informed the exception will not be handled by the associated method.

An exception is handled by using a **try-catch block**. The compiler is informed that an exception will not be handled by a method by declaring the exception in a method’s throws clause. We then say the method throws the exception:

**The syntax of declaring an Exception:**

**methodType methodName(parameters) throws ExceptionType1, ExceptionType2, . . ., ExceptionTypeN {**

**...**

**}**

Example:

**public static void main(String[] ags) throws FileNotFoundException {**

**// A statement that may cause FileNotFoundException**

**. . .**

**}**

The syntax of try-catch block that is used to catch an exception:

**try{**

**. . .**

**statementThatMayCauseException;**

**. . .**

**} catch(ExceptionType e){**

**Code to handle the exception**

**}**

If an exception occurs, the statements in the catch block that follow the statement that caused the exception are not executed; control is transferred directly to the catch block. If no exception occurs, all statements in the catch block are executed and the statements in the catch block are not executed.

Example:

**try{**

**// A statement that may cause FileNotFoundException**

**. . .**

**} catch(FileNotFoundException e){**

**System.out.err(e);**

**System.exit(1);**

**}**

**Note:** If you do *catch* an exception, do not add it to the *throws* clause in the method header.

**File Input**

**Steps for text-file input:**

1. Use a **FileInputStream** object to create a **Scanner** object. Since the **Scanner** class and **FileInputStream** class are not in the **java.lang** package, import them by adding one of the following statements at the top of your program:

**import java.util.Scanner;** // or **import java.util.\*;**

**import java.io.FileInputStream; // or import java.io.\*;**

**FileInputStream instream = new FileInputStream( fileNameString );**

**Scanner inFile = new Scanner( instream );**

Note: These statements may be combined:

**Scanner inFile = new Scanner(new FileInputStream(fileNameString));**

The statement **new FileInputStream( fileNameString );** may throw a checked exception **FileNotFoundException**; you must either throw the exception or catch it. Whatever exception class is used, it must be imported from java.io package. Example:

|  |
| --- |
| **import java.util.\*;**  **import java.io.\*;**  **public class MyFileIO{**  **public static void main(String[] args) throws IOException {**  **FileInputStream instream**  **= new FileInputStream("D:\\Workarea\\input.txt");**  **Scanner inFile = new Scanner(instream);**  **// ...**  **}**  **}** |
| **import java.util.\*;**  **import java.io.\*;**  **public class MyFileIO{**  **public static void main(String[] args){**  **FileInputStream instream**  **= new FileInputStream("D:\\Workarea\\input.txt");**  **try{**  **Scanner inFile = new Scanner(instream);**  **// ...**  **} catch(FileNotFoundException e){**  **System.out.err(e);**  **System.exit(1); // terminate the program**  **}**  **// . . .**  **}**  **}** |

**Note:**

* If the text-file to be read is in the same folder as the Java program, the full path need not be specified. Example:

**"input.txt"**

* Single forward-slashes or double back-slashes may be used for the file path. Examples:

**"D:/Workarea/input.txt"**

**"D:\\Workarea\\input.txt"**

* The file path may be read from the console. Example:

**System.out.print("Enter the file name: ");**

**String fileName = scanner.nextLine();**

**FileInputStream instream = new FileInputStream( fileName );**

**Scanner inFile = new Scanner( instream );**

1. Use an appropriate **Scanner** object method to read the input from the file. Example:

**double speed = inFile.nextDouble( );**

The **next** methods of a scanner object that is connected to a file behave like the scanner object connected to **System.in** with the addition that if an attempt is made to read and there are no more tokens in the input file, a **NoSuchElementException** is thrown.

1. Do the required processing of the data read from the file.
2. When you have finished reading from the file, close the input stream. Example

**inFile.close();**

Closing a stream when it is no longer needed is very important. This practice helps avoid serious resource leaks.

**Example:**

|  |
| --- |
| import java.io.FileInputStream; import java.io.FileNotFoundException; import java.util.Scanner; //Finds max of three numbers read from input file numbers.txt public class FileRead{  public static void main(String[] args)throws FileNotFoundException{  double num1, num2;   int num3;  FileInputStream instream = new FileInputStream("numbers.txt");  Scanner myScanner = new Scanner(instream);  num1 = myScanner.nextDouble();  num2 = myScanner.nextDouble();  num3 = myScanner.nextInt();  System.out.printf("Max = %.2f%n", Math.max(num1, Math.max(num2, num3)));  myScanner.close();  } } |

**Reading one character at a time from a file**

Java's Scanner class does not have a built in method to read character-by-character. Each of the following techniques may be used to read a **string** and then extract the first character of that string:

**Scanner inFile = new Scanner(instream);  
char character = inFile.next().charAt(0);**

or

**char character = inFile.nextLine().charAt(0);**

To read exactly one character use the **read** method of a FileInputStream:

Example:

|  |
| --- |
| **import java.io.\*;**  **public class FileCharRead {**  **public static void main(String[] args){**  **try{**  **FileInputStream fis = new FileInputStream("data.txt");**  **char ch = (char) fis.read();**  **System.out.println("The character read from the file is " + ch);**  **fis.close();**  **}catch(IOException e){**  **System.err.println(e);**  **}**  **}**  **}** |

**Note:** The read method returns the Unicode of the character read. It also throws IOException.

An alternative way of reading exactly one character use a **Scanner** object’s **next** method but specify the input delimiters [or token separators] as the empty string "":

**FileInputStream instream = new FileInputStream("input.txt");  
Scanner fileScanner = new Scanner(instream);**

**fileScanner.useDelimiter("");**

**char ch = fileScanner.next().charAt(0);**

**Writing to a File**

To write to a file, the print( ), println( ), or printf( ) methods of the **PrintWriter** class may be used. The **PrintWriter** class is defined in the **java.io** package. Some constructors of the PrintWriter class are:

[PrintWriter](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\io\PrintWriter.html#PrintWriter(java.lang.String))([String](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\String.html) fileName)

[PrintWriter](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\io\PrintWriter.html#PrintWriter(java.io.OutputStream))(File[OutputStream](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\io\OutputStream.html) out)

Example:

|  |
| --- |
| import java.io.FileOutputStream;  import java.io.PrintWriter;  import java.io.IOException;  public class FileWrite {  public static void main(String[] args) throws IOException {  FileOutputStream outstream = new FileOutputStream("Example.txt");  PrintWriter pwriter = new PrintWriter(outstream);  pwriter.println("Riyadh");  pwriter.println("Abha");  pwriter.println(22.85);  pwriter.println('A');  pwriter.printf("Square root of 16.567 is %.2f%n", Math.sqrt(16.567));  pwriter.print(true);  pwriter.close();  // For convenience, let the user know that the file has been created  System.out.println("The file has been created.");  }  } |

**Note:** If a file is opened for writing and it does not exist, it is created. If it exists, its current contents are erased.

**Appending data to a text file**

1. Create a **java.io.FileOutputStream** object using the constructor:

**FileOutputStream(String filename, boolean append)**

with the append parameter set to **true**.

1. Wrap the **FileOutputStream** object in a **PrintWriter** object.
2. Use **print, println,** or **printf** methods of the **PrintWriter** object.
3. Close the output stream associated with the **PrintWriter** object.

|  |
| --- |
| import java.io.\*;  public class FileAppend {  public static void main(String[] args) throws FileNotFoundException {  FileOutputStream outstream = new FileOutputStream("Example.txt", true);  PrintWriter pwriter = new PrintWriter(outstream);  pwriter.println("Jubail");  pwriter.println("Hafr Al-Batin");  pwriter.println(22.85);  pwriter.println('B');  pwriter.print(false);  pwriter.close();  // For convenience, let the user know contents appended to file  System.out.println("Contents appended to file.");  }  } |

**Note:** A file that is opened for appending is created if it does not exist.

* **Using loops in Text-File I/O**

The following **Scanner** boolean methods may be used in loops that read up to the end of a file:

|  |  |
| --- | --- |
| Method | Effect |
| boolean hasNext() | Returns true if this scanner has another token in its input. The scanner does not advance past any input. |
| boolean hasNextLine() | Returns true if there is another line in the input of this scanner. The scanner does not advance past any input. |
| boolean hasNextInt() | Returns true if the next token in this scanner's input can be interpreted as an int value. The scanner does not advance past any input. |
| boolean hasNextLong() | Returns true if the next token in this scanner's input can be interpreted as a long value. The scanner does not advance past any input. |
| boolean hasNextFloat() | Returns true if the next token in this scanner's input can be interpreted as a float value. The scanner does not advance past any input. |
| boolean hasNextDouble() | Returns true if the next token in this scanner's input can be interpreted as a double value. The scanner does not advance past any input. |

Example: The following Java program fragment computes the average of grades in a file associated with fscanner and outputs the average to a file associated with pwriter:

**// . .**

**int count = 0;**

**double grade, sumOfGrades = 0;**

**while(fscanner.hasNextDouble()){**

**grade = fscanner.nextDouble();**

**count++;**

**sumOfGrades += grade;**

**}**

**if(count != 0)**

**pwriter.printf("Average = %.2f%n", sumOfGrades / count);**

**else**

**System.out.println("Error: No grades read");**

**fscanner.close();**

**pwriter.close();**

* **Reading one character at a time up to the end of a text-file**

When the **read** method tries to read beyond the end of file, it returns an integer -1. We can use this -1 as a sentinel in a loop that reads one character at a time from a file.

Example: Write a Java program that writes to a file **output.txt** a copy of a file **input.txt** in which all lowercase characters are converted to uppercase:

**import java.io.\*;**

**public class FileReadChar2{**

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

**FileInputStream fis = new FileInputStream("input.txt");**

**FileOutputStream fos = new FileOutputStream"output.txt");**

**PrintWriter pwriter = new PrintWriter(fos);**

**int unicode, count = 0;**

**char ch;**

**while((unicode = fis.read()) != -1){**

**ch = (char) unicode;**

**ch = Character.toUpperCase(ch);**

**pwriter.print(ch);**

**}**

**fis.close();**

**pwriter.close();**

**System.out.println("The file output.txt has been created . . . ");**

**}**

**}**

Note: The above example can be solved using a Scanner:

**import java.io.\*;**

**import java.util.Scanner;**

**public class FileReadChar3{**

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

**FileInputStream instream = new FileInputStream("input.txt");**

**Scanner fscanner = new Scanner(instream);**

**FileInputStream outstream = new FileInputStream("output.txt");**

**PrintWriter pwriter = new PrintWriter(outstream);**

**int count = 0;**

**char ch;**

**fscanner.useDelimiter("");**

**while(fscanner.hasNext()){**

**ch = fscanner.next().charAt(0);**

**ch = Character.toUpperCase(ch);**

**pwriter.print(ch);**

**}**

**fscanner.close();**

**pwriter.close();**

**System.out.println("The file output.txt has been created . . . ");**

**}**

**}**

* **Using String Scanner in Text-File I/O**

A scanner can read items from a string. Example:

**String str1 = "Ayoub Saleem 5.0 3.0 7.5";**

**Scanner stringScanner = new Scanner(str1);**

**System.out.print("The total score of ");**

**System.out.print(stringScanner.next());**

**System.out.print(" " + stringScanner.next() + " is: ");**

**double sum = 0;**

**while(stringScanner.hasNextDouble())**

**sum += stringScanner.nextDouble();**

**System.out.println(sum);**

**stringScanner.close();**

The use of String scanner is required in some Text-File I/O problems. For example, obtaining student grade totals from a text-file of the form:

920000045 Said Fadhel 50.0 70.0

920000046 Muhammad Omar 70.5 80.0 78.0 90.0

920000047 Zaid Khan 90.0 85.0 79.5

920000048 Yusuf Muhsin 75.0 80.0

920000049 Ahmad Abdallah 70.5 80.0 78.0 90.0 100.0

920000050 Adel Qasim 40.0 50.0 60.5

920000051 Muneeb Muneer 85.0 90.0

In this case the following nested loops may be used:

**String line, name1, name2;**

**int id;**

**double sum;**

**Scanner stringScanner;**

**while(fileScanner.hasNextLine()){**

**sum = 0;**

**line = fileScanner.nextLine();**

**stringScanner = new Scanner(line);**

**id = stringScanner.nextInt();**

**name1 = stringScanner.next();**

**name2 = stringScanner.next();**

**while(stringScanner.hasNextDouble())**

**sum += stringScanner.nextDouble();**

**// . . .**

**}**

**fscanner.close();**

**stringScanner.close();**

**Laboratory Tasks**

**Task 1:** Write a Java program that reads the surface area and radius of a cylinder from a file **input.txt**, it then calculates and writes the volume of the cylinder to a text-file **output.txt**. The volume must be written with an appropriate message and in two decimal places.

|  |  |
| --- | --- |
| Cylinderr2.jpg |  |

Sample inputs and outputs:

|  |  |
| --- | --- |
| input | output |
| 175.93 2.0 | Volume = 150.80 cubic cm |
| 400.0 4.5 | Volume = 613.72 cubic cm |

**Task 2:** Write a Java program that reads an input text-file it then writes to an output text-file the number of uppercase characters, lowercase characters, digit characters, white space characters, other characters, and the total number of characters in the text-file.

**Note**: In windows, each line of a text-file, except the last line, is terminated by the character pair **"\r\n"**

Sample input file:

|  |
| --- |
| This is  a Test File.  456 7 |

Sample output file:

|  |
| --- |
| Number of uppercase characters = 3  Number of lowercase characters = 12  Number of digit characters = 4  Number of whitespace characters = 8  Number of other characters = 1  Total number of characters = 28 |

**Task 3:** Write a Java program that copies the given input file **arabic.txt** to an output text-file.

Hint:

* Create a scanner object that reads from a UTF-8 encoded text-file by:

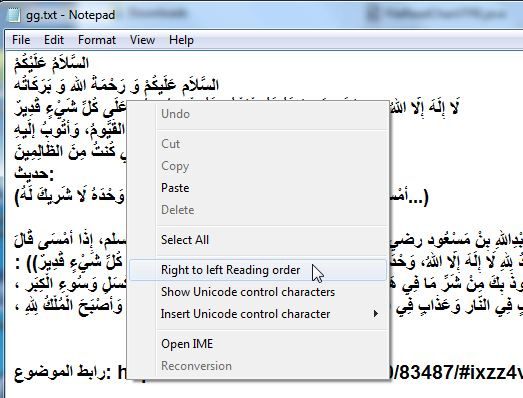
**FileInputStream instream = new FileInputStream("arabic.txt");**

**Scanner fileScanner = new Scanner(instream, "UTF-8");**

* Create a **PrintWriter** object that creates a UTF-8 encoded text-file by:

**PrintWriter pwriter = new PrintWriter("output.txt","UTF-8");**

The created file will have left to right text-direction, to view the text in right to left direction, open the file using Notepad, right click on Notepad window and click **Right to left Reading order**:



**Task 4:** An input-file **scores.txt** contains an unknown number of lines. Each line in the file contains two values: ID# and the score of a student in a certain exam.

Write a Java program that first uses a scanner to read the input file to compute the average score of the students, it then closes the scanner, then opens the file again for reading and finally it reads the file a second time to distribute the students into two output files, **good.txt** containing those students whose scores are greater or equal to the average, and **poor.txt**, containing those students who scored less than the average.

|  |  |  |
| --- | --- | --- |
| Input file scores.txt: | Output file good.txt: | Output file poor.txt: |
| 206527 44.24  208530 75.38  207135 85.61  205241 91.51  204324 50.61  203357 68.28  202117 57.11 | ID SCORE  208530 75.38  207135 85.61  205241 91.51  203357 68.28 | ID SCORE  206527 44.24  204324 50.61  202117 57.11 |

**Task 5:** Write a Java program that prompts for and reads a word. It then finds and prints on the monitor the frequency of the word in a text file **input.txt**.

Note: To use a file with punctuation characters use the **useDelimiter** method of Scanner:

**FileInputStream instream = new FileInputStream("input.txt");**

**Scanner fileScanner = new Scanner(instream);**

**fscanner.useDelimiter("[ \n\t\r?:.,;!]+");**

You may use the following sample input file:

test this;

this. here this. this.

rest this here this