CSCI 221: Computer Programming II **Due, Tuesday, Nov. 21, 5pm**

Programming Assignment 6 Total Points 100

Collaboration Policy: None

Nadia has a To-Do list. She needs to know which items she still needs to do. Your job is to create a program that reads a file that contains Nadia's To-Do list [figure 1.1] and outputs only the ToDo items to the console [figure 1.2]. Unfortunately, Nadia has many different To-Do lists and there may be some syntax problems. You must read the file line-by-line and check the validity of the file by catching exceptions with Try-catch blocks. **Please note, do not use any code found on the web to complete this assignment. You may look at the java API, Oracle java tutorials, and your java textbook.**

You must develop a SymbolException class, a KeywordException class, and a CheckLine class whose API specification are provided below. Make sure the names of your packages, classes, and methods match those provided below **exactly**, including capitalization. The number of parameters and their order must also match. Otherwise, your implementations will not work correctly and my test cases will fail.

Additionally, a ReadLogFile.java file (that you must complete – see ReadLogFile class below) and three test files have been provided to you (although I will test your program with other files, that meet the format described next). For each test file you may assume the format of each line is: a keyword, a single space, and then a description. The keyword will only be a single word (i.e. no spaces) and the description may be any String value. Also, you may assume no empty lines exist between lines or after the last line.

SymbolException Class

The SymbolException class must **inherit** from the java Exception class. The API specification for the SymbolException class is provided below.

Constructors	**Description and Preconditions (if any)**
SymbolException ()	A **public** no argument constructor Call the SymbolException(String message) constructor with the string **"**Default Message**"** (Hint: remember java's keyword this) Precondition: None.

SymbolException(String message)	A **public** single argument constructor. Remember you must call the superclass's constructor in order to set the message variable. Precondition: None.

KeywordException Class

The KeywordException class must **inherit** from the java Exception class. The API specification for the KeywordException class is provided below.

Constructors	**Description and Preconditions (if any)**
KeywordException ()	A **public** no argument constructor. The KeywordException(String message) constructor is called with the string **"**Keyword Not Found**"** Precondition: None.
KeywordException(String message)	A **private** single argument constructor. Making the constructor private ensures the class cannot be instantiated through this constructor. Remember you must call the superclass's constructor in order to set the message variable. Precondition: None.

CheckLine Class

The API specification for the CheckLine class is provided below. When exceptions are thrown be sure to pass a message that is detailed.

Private Variables	**Description and Preconditions (if any)**
private String [] keywords;	This variable will hold an array of keywords
private char [] invalidSymbols;	This variable will hold an array of invalid symbols

public Constructors	**Description and Preconditions (if any)**
CheckLine(String []kWords, char [] invalidS)	A <u>**public**</u> two argument constructor. Parameters: a String array and a char array In this constructor initialize the keywords and invalidSymbols arrays with the arrays passed into the constructor. Precondition: kWords != null and invalidS != null

public Methods	**Description and Preconditions (if any)**
checkFirstKeyword(String strKeyword)	This method **returns** a boolean value. Precondition: strKeyword != null Returns true if the provided string matches the **first element** in the keywords array. Returns false in any other circumstance.
checkForInvalidSymbols (String line)	The provided string is tested to make sure there are no invalid symbols found. Precondition: line != null If the provided string has an invalid symbol from the invalidSymbols array, then the exception SymbolException (with a detailed message) will be thrown for the <u>**first found**</u> invalid symbol, else do nothing. [see figure 2]
checkForInvalidKeyword(String strKeyword)	The provided string is tested to see if it matches a keyword in the keywords array. Precondition: strKeyword != null If the string is not a keyword found in the keywords array, throw a KeywordException with no parameters, else do nothing. [see figure 3]

ReadLogFile class

1. Download the ReadLogFile.java from the dropbox.This is the driver for your solution.
2. Look at the code in ReadLogFile and notice:

    a. There is a section of code that tells you not to change it, so **<u>do NOT</u>** change it. (i.e. do not change the elements of the arrays or the names of the arrays) **<u>Do not</u>** make direct calls to the elements of the arrays. The only time you use these variables are in #4d.
    b. There is a **String path** given to you. You must use this string to specify the directory path to a test file, so you can test your code. This variable will come in handy for #4. (see pg 757)
3. Add code to ReadLogFile:
    a. Use a Scanner object and File object to read from a file. To specify the path, use the variable **path**. (see chp 10 and Lab6). To read from a files we had to:
        i. Import the java io file package:
            import java.io.File;
        ii. Create a File instance to reference text file in Java and specify the path. You will use the variable you created in 3b above as the argument here
            File text = new File(<path to the file>);
        iii. Create a Scanner instnace to read File in Java
            Scanner scan = new Scanner(text);
    b. Make sure you use a Try-catch statement to catch any Exceptions.
    c. **<u>Create a String variable</u>** to hold a line from the file being read.
    d. **<u>Create a CheckLine</u>** object with parameters keywords and invalidSymbols.
4. For each line of the file you **<u>must call in order the methods</u>**:
    a. checkForInvalidSymbols
    b. checkForInvalidKeyword
    c. checkFirstKeyword
5. You must print each line in the To-Do list if the Keyword matches the first keyword in the keywords array.
6. Feel free to use loops of your choosing.
7. Remember that **all** Exceptions should be caught using catch blocks and exception messages must be printed to the console. (This includes the FileNotFound exception that the File methods might throw.)
8. Your driver may terminate immediately after it detects an exception and generates an appropriate message.

Additional Specifications:

- Exception handling (i.e. try/catch blocks) **<u>should</u>** be included in this assignment

***__Hints:__**

- Start by reading and printing out each line of file1.txt, because if your program is not reading the file, nothing else will work.
- Start on checkForInvalidSymbol only after getting other methods to work
- Try printing out the length of each line of the file

- Use badFile1.txt to make other bad files so you can test your code.
- Look at the API for String methods

Figure 1.1: file1.txt

```
DONE buy apples
ToDo buy milk
DONE buy bread
ToDo buy bottled water
ToDo buy coffee
DONE buy tea
ToDo Fix the sink
```

Figure 1.2: file1.txt output

```
ToDo buy milk
ToDo buy bottled water
ToDo buy coffee
ToDo Fix the sink
```

Figure 2: badfile1.txt example output

```
ToDo buy milk
KeywordException: edu.cofc.csci221.KeywordException: **Keyword Not Found**
```

Figure 3: badfile2.txt example output [Notice that the exception is for the first invalid symbol in the file.]

```
ToDo buy milk*
SymbolException: There is an invalid symbol [!]
```

Program Documentation

Refer to the **documentation standard posted in the content section on OAKS**. Each file you turn in must have a comment header with your name, date, and class description. Each method MUST be preceded by an appropriate Java doc comment. Use Eclipse short cuts (e.g. /** - return key) to generate comment stubs above class definition and methods.

For each method in the CheckLine class (in the comment block above the method) state the pre- and post-conditions, i.e. what is expected before and after the method is called (see page 300 in the course textbook).

Program Submission

Create a ZIP file that only contains the completed CheckLine, ReadLogFile, KeywordException and SymbolException java files, that is, no byte-code files (i.e. .class files) are to be submitted. Furthermore, please do not include a file hierarchy (i.e. the package structure – package specification line). For the ZIP file you **must** use the naming convention: *<lastname>*.zip

<u>If the assignment is not submitted in the correct format via OAKS – it will not be accepted – no exceptions!</u> Submit the ZIP file via OAKS in the Dropbox that corresponds to the assignment. Resubmit, as many times as you like, the newest submission will be the graded submission.

Grading Rubric

20 Points	Style: Comments/Indentation
80 Points	Functionality: • Your Program compiles/runs (10) • Test Cases (70) o test files o And other test cases

If the submitted program does not compile: 0 of 80 points
If the submitted program compiles but does not run: 5 of 80 points
If the submitted program compiles and runs: 10 of 80 points
If the submitted program compiles, runs, and produces correct output: 80 of 80 points

Late assignments will not be accepted – no exceptions (please do not email me your assignment after the due date, I will not accept it).

Please feel free to setup an appointment to discuss the assigned problem. I'll be more than happy to listen to your approach and make suggestions. However, I cannot tell you how to code the solution. Furthermore, code debugging is your job; please do not come to my office asking me to fix your code.