CS 1110, LAB 8: EXCEPTIONS

Name: _____ **Net-ID**: _____

There is an online version of these instructions at

http://www.cs.cornell.edu/courses/cs1110/2012sp/labs/lab08.php

You may wish to use that version of the instructions.

The purpose of this lab is to give you some experience with writing recursive functions. It is designed to be reasonably short so that you do not have to worry about working on this assignment over Spring Break.

Requirements For This Lab. The very first thing that you should do in this lab is to download the file `ExceptionHandling.java` from the course web page:

http://www.cs.cornell.edu/courses/cs1110/2012sp/labs/lab08/ExceptionHandling.java

You should create a new directory on your hard drive for this file. This file contains a mixture of function stubs and some (mostly) completed functions and procedures. Your assignment is to answer questions about the functions/procedures that are completed, and to complete those that are not.

For this lab you will show your instructor the contents of `ExceptionHandling.java` and what you have written on this sheet. In addition, you are asked to write a new (simple) class called `MyException.java` from scratch; you should show that to your instructor as well. As always, you should try to finish the lab during your section. You can show it at the next lab after Spring Break, but we recommend that you do not leave yourself work to do over Spring Break.

If you get stuck, do not waste time. Ask the TA or consultant for help.

---

1. Exceptions and User Input

Exceptions are very common when handling user input from a keyboard. You have absolutely no idea what the user will type into the keyboard, and so you have to be prepared for anything.

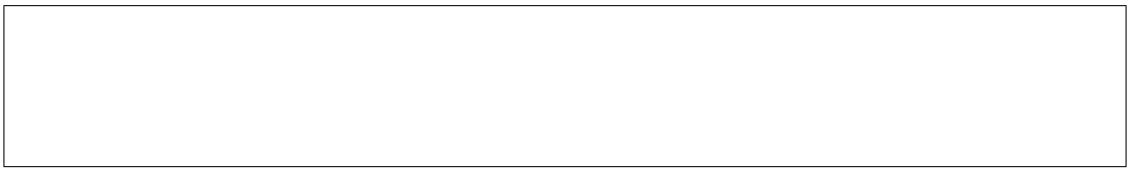
1.1. **The Function `getKeyboard()`.** You will be using this function for this part of the lab. The specification for this function says the following:

```
/** Yields: a buffered reader attached to the keyboard. If you store
 *  this value in a variable such as kbd, then
 *
 *      kbd.readLine()
 *
 *  will prompt the reader to type a line by putting in the interactions
 *  pane a field into which to type and will then yield the string of
 *  characters the user typed in.
 */
```

Study the contents of this function. Then try it out by typing the following into the the Interactions pane:

```
BufferedReader c = Lab12. getKeyboard();
c.readLine()
```

When the box appears in the Interactions pane, type something into it and hit the return/enter key. What happens when you do this?



1.2. **Checked Exceptions.** The specification for `readKeyboardLine()` is as follows:

```
/** Prompt the reader to type a line into the interactions pane and
 *  return the line that the user types (e.g. this is function with
 *  side-effects).
 *  Yields: The text the user typed into the interactions pane
 *  Throws: IOException if there is a hardware issue
 */
```

There are several interesting thing to note about this function. First of all, it is a function with "side-effects", as we talked about in class. So, the specification says both what it does and what value it returns. Second, this is a function that throws an exception that must be checked; we include this information in the specification.

Study the contents of this function. Note that the exception thrown is an instance of `IOException`. IO stands for input-output, and an `IOException` is thrown whenever some sort of input-output error happens. Delete the clause `throws IOException` from the header of function `readKeyboardLine()` and try to compile. What happens?

Afterwards, put the throws-clause back in the function header and compile.

As explained on pp. 323-325 of the text, Java expects that an exception that may be thrown in a method body is either (1) caught in that method body or (2) mentioned in the throws clause of the method header; the latter is so that the user has aome sort of indication that the exception may be thrown. All exceptions that may be thrown must be checked except for instances of `RuntimeException` (and its subclasses).

Do not actively be concerned about the throws clause. Instead, work as follows. Whenever you try to compile but Java says that a throws clause is needed, put it in.

1.3. **The function `readKeyboardInt()`.** The function (with side-effects) `readKeyboardInt()` has the following specification:

```
/** Prompt the reader to type an integer (an int) into the interactions pane
 *  and return the integer that they type. If the user types something that
 *  is not an int, then issue a message (System.out.println(...) and prompt
 *  again (HINT: Use recursion).
 *  Yields: the number the user typed into the interactions pane
 *  Throws: IOException if there is a hardware issue or if the user
 *  did not type a number.
 */
```

As you can see, it is supposed to keep prompting the user for an integer, giving them a message when they type something other than an integer. As we said in the hint, you should do this with a recursion in a try-catch block. In other words

```
try {
    // try to read an integer, base case if successful
} catch (decl. of an exception variable) {
    // error message, plus recursive call to try again
}
```

To convert user input to an int, you should use the static method `Integer.parseInt(String)`. Note that this method has the following specification (editted from the Java API):

```
/** Parse s as a signed decimal integer. The chars in s must be decimal digits, except that
 *  the first character may be an ASCII minus sign '-' to indicate a negative value.
 *  Yields: the resulting integer value.
 *  Throws: a NumberFormatException if s does not contain a parseable int.
 */
```

Therefore, `NumberFormatException` is the exception that you should be prepared to catch.

---

2. Exceptions and Preconditions

Throughout this course, we have used the term precondition for a true-false statement that should be true when the method is called, and it is the caller's duty to ensure that it is true. For example, look at the function `exp(double,int)` in `ExceptionHandling`. It has the precondition $c \geq 0$. In this case, if the user calls exp with $c < 0$, infinite recursion results.

Excepion handling can be used to enable more robust programming. A robust program will prevent abnormal termination and unexpected actions like this infinite loop. Robust programming requires that invalid inputs, such as invalid arguments of a call, be handled in a reasonable way. In the case of function `exp`, the function itself does not know how to handle the error $c < 0$, but it can throw an exception so that the user can handle it.

2.1. **Make `exp` Robust.** Change the specification of function exp so that it says that it throws an exception if c ¡ 0, indicating which one is thrown. You can choose an appropriate one from the list below; these (and many more) are defined in the Java API. Then, change the function itself to throw that exception, with a suitable message in the exception, if $c < 0$.

Exception	Description
`ArithmeticException`	Thrown when an exceptional arithmetic condition has occurred. For example, an integer "divide by zero" throws an instance of this class.
`FileNotFoundException`	Thrown when an attempt to open the file denoted by a specified pathname has failed.
`IndexOutOfBoundsException`	Thrown to indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.
`IllegalArgumentException`	Thrown to indicate that a method has been passed an illegal or inappropriate argument.
`IllegalStateException`	Thrown when a method has been invoked at an illegal or inappropriate time. In other words, the Java environment or Java application is not in an appropriate state for the requested operation.
`UnsupportedOperationException`	Thrown to indicate that the requested operation is not supported.

Make sure function exp works correctly by using the function call `ExceptionHandling.exp(1,-3)` in the Interactions pane. Write down what happens here:

2.2. **Write a Throwable Class Definition.** Define the subclass `MyException` of class `Exception`. It needs two constructors: one with no parameters and one with one String parameter, which is the detail message. Be sure to write specifications for the constructors.

You should not add any new fields or methods. The only thing you need to subclass `Exception` is the two constructors.

2.3. **Test out `MyException`.** Test what you have done using the function `factorial` in `ExceptionHandling`. Like `exp`, this method also has the precondition `n` > 0. You are to change this function so that it throws a `MyException` instance if it receives a value of `n` ¡ 0.

First add the exception to the method body **without changing the method header**. That is, leave the following part alone:

```
public static int factorial(int n) {
```

Compile your program. What happens?

Now change the header above so that your code properly compiles. When you are done, you are finished with the lab.