Computer Science 241

Lab 4 (10 points)
Due Sunday, May 6th, 2018 at 10:00 PM

Read all of the instructions. Late work will not be accepted.

Overview

This week's lab focuses on Java *exceptions*. Exceptions in Java are objects that get "thrown" when some exceptional circumstance is encountered; for example, if you ask it to read from a file that does not exist. There are three broad kinds of exceptions

1. Errors: these happen when something has gone seriously wrong, external to your program (for example, you're not able to access a disk). You (the Java programmer) are not required to handle these, since recovering from such errors can be very difficult.
2. Runtime Exceptions: these happen when something is mismatched at run-time that indicates a programming bug, as opposed to a bad user input. Examples include passing a reference variable with null when it is assumed the reference variable points to an object, dividing by zero, or attempting to access an index out of an array's range. You can handle these, but are not required to.
3. Checked exceptions. All other exceptions are known as "checked exceptions." It is assumed that you can and should handle such exceptions. These include errors that might indicate bad user input, such as a java.io.FileNotFoundException. You have to either handle, or explicitly acknowledge that you don't handle, checked exceptions.

If you call a method that can throw a checked exception, you must do something with it. The easiest way to do so is to modify your method's header to indicate that it can itself throw the exception. This just passes the buck to whatever method called yours. For example:

```
public static void xyz(String filename) throws java.io.FileNotFoundException {
    // implementation here
}
```

After "throws" you can provide a comma-separated list of the exceptions that can be thrown by methods your method calls, but which you don't explicitly handle in your method. To explicitly handle an exception, you need try and catch blocks. The syntax is illustrated below:

```
try {
    output = new java.io.PrintStream("outputFile.txt");
} catch( java.io.FileNotFoundException e ) {
    // Here we "handle" the FileNotFoundException exception
    // (e.g. by prompting the user to give a valid output file)
    // this code only runs if an exception was thrown by PrintStream
    // You can have multiple catch blocks if there are multiple exceptions
    // that can be thrown in the try block
    System.err.println("FileNotFoundException exception!");
```

```
    System.exit(1);
}
```

Lab Description

Start with the `ExceptionLab.java` file I provided for this lab. Below are a series of steps (S1, S2, ...) and interleaved questions (Q1, Q2, ...). Please complete them all, and write your answers to the questions in a file named `writeup.txt` that you will submit (see Submission instructions later in this pdf).

S1 Compile `ExceptionLab.java`.

Q1 Run `java ExceptionLab 1 2 3 5 5 6 7 8 9`. What exception is thrown? What line of `ExceptionLab.java` throws it?

Q2 Run `java ExceptionLab 1 2 3 4 5`. What exception is thrown? What line of `ExceptionLab.java` throws it?

Q3 Run `java ExceptionLab 0 1 2 3 4 5 6 7 8 9`. What exception is thrown? What line of `ExceptionLab.java` throws it?

Q4 Run `java ExceptionLab 1 2 hi 4 5 6 7 8 9`. What exception is thrown? What line of `ExceptionLab.java` throws it?

Q5 Uncomment `writeArrayToFile(x,".tmp.lab5.txt");` and the `writeArrayToFile` method and recompile. What compilation error message do you get?

S2 Fix the `writeArrayToFile` compilation errors by adding the appropriate `throws` expression to the method header and wrapping the call to `writeArrayToFile` in a try block. Its corresponding catch block should print to an error message to standard out. The error message should consist of "`writeArrayToFile threw exception: `" followed by the string representation of the exception object (hint: you can just call print on the exception and it will automatically invoke the object's `toString` method).

S3 In the `main` method, wrap the calls to both `makeArray` and `identity` in a single try block. This try block should be followed by two catch blocks: one that catches the exception you triggered in `makeArray` and another that catches the exception you triggered in `identity`. These two catch blocks should print a statement to standard error of the same form as in S2, replace `writeArrayToFile` with `makeArray` and `identity`, respectively. In those catch blocks, after the print to standard error, you should call `System.exit(1)`, since as written we cannot recover from errors in either of those methods. *Note: if you declare* `x` *in the try block, it will be out of scope outside of the try block. This is a common scenario. One tip is to declare* `x` *before the try block, setting it to* `null` *temporarily, and then within the try block assign the result of* `makeArray` *to* `x`.

S4 In the `main` method, wrap the call to `slice` in a try block, whose catch block just prints a message to standard error following the pattern of the previous questions. This catch block should *not* exit - we want to continue running the subsequent code even if `slice` throws an exception.

S5 In the `main` method, wrap the call to `buggierSlice` in a try block, whose catch block just prints a message to standard error following the pattern of the previous questions. This catch block should *not* exit - we want to continue running the subsequent code even if `buggierSlice` throws an exception.

Q6 Read up on the `finally` block, which can be used in conjunction with `try` and `catch` blocks. In your own works, when and why might it be useful?

Submission

Once you have finished all of the above, `add`, `commit` and `push` the following files to your course repository `master` branch:

- `lab5/ExceptionLab.java` (after your modifications)
- `lab5/writeup.txt`

Note that these two files should be directly within a `lab5` subdirectory of your repository (spelling, spacing and capitalization matter). As always, you can confirm that your code is properly submitted by checking your github repo URL:

`https://github.com/hutchteaching/201820_csci241_username`

Grading

At the deadline a script will automatically clone your repository. Points will be deducted for any problems in your submission, including:

- Missing or incorrectly named files or directories
- Failing to complete all of the walk-through steps.
- Incomplete or incorrect answers to the questions.
- Spelling or grammatical errors in `writeup.txt`.
- Managing to introducing poor coding style into `ExceptionLab.java`
- Other assorted failures to follow Lab 5 instructions