CS212: Paradigms
Dr. Gross, Spring 2016

Lab 6 - Exceptions and File I/O (no clever name this week)

1. This one time, no need to write OO code; just use static methods
2. Create a new package called edu.blackburn.cs.cs212sp16.fileio.lastnamelastname
3. Make a Runner in this package
4. Create a main() method

Newb Mode:

The basic code for a try/catch block looks like this:

```
// stuff before Exception-throwing code
try {
      // Exception-throwing code
} catch (Exception e) {
      // what to do if and only if an exception is thrown
      // for now, we'll just be printing out the exception
}
// stuff that happens after the exception throwing code
```

1. Create a new a static method called testDivideByZero()
2. In that method, write code to print "Divide By Zero Checkpoint 1"
3. Write a try catch/block below that and print the exception in the catch block
4. Inside the try block divide by zero
5. Print "Divide By Zero Checkpoint 2" inside the try block but after the code above
6. Print "Divide By Zero Checkpoint 3" below the try/catch block
7. Print a blank line
8. Call the method from main()
9. What happens when you run this? Why?

Easy Mode:

Let's open a file.

1. Create a text file **in your package** called "lab06.txt" and put some text in it (multiple lines)
2. Add and commit files
3. Create a new static method in Runner called openFile(String filename)
4. In that method, print "Opening File: " + filename
5. Print "Open/Read Checkpoint 1"
6. Create a try/catch block and print the exception in the catch block
7. In the try block
    a. Open the file using a new FileReader instance
    b. Pass that object to a new BufferedReader
    c. Print "Open/Read Checkpoint 2" after the code above

8. In the catch block, write code to print the exception
9. Print "Open/Read Checkpoint 4" (yes, 4) below the try/catch block
10. Print a blank line
11. Call the close() method on the BufferedReader instance
12. Call the method from main(), passing in the correct filename
13. What's the correct format for the filename?
14. Should you use relative or absolute paths? Why?
15. Call the method from main again, but this time with an incorrect filename (two separate calls)

Hard Mode:

Polymorphism fun! All exceptions are subclasses of the appropriately named Exception class. When we catch an exception, we can either catch all exceptions (using Exception as the type) or we can catch a specific subclass of Exception. We do this to make different code to handle different problems. You can find out what exceptions a method can throw in the Java API.

1. Make a copy of the openFile() method called readFile()
2. Look up the correct Exception subclass that FileReader's constructor can throw
3. In readFile()'s catch block, change the exception type to that specific type
4. In the catch block, don't print the exception anymore, just print "No such file"
5. Run this and see what happens
6. Now create a loop to read in all lines and print them out (remember the code in the lecture repo)
7. Print "Open/Read Checkpoint 3" after the code above (still in the second catch block)
8. Compile and… wait, it doesn't compile! We've broken Java; you can't ignore (most) Exceptions
9. Look up the new exception we have to handle when calling readLine() and read below

You can have as many different catch blocks as you need. They need to go from more specialized (subclasses at the bottom of the hierarchy) to more generalized (classes closer and closer to the parent Exception) because polymorphism. Let's assume we have three exceptions:

- HighestException, which extends Exception
- MiddleException, which extends HighestException
- LowestException, which extends MiddleException

The code looks like this:

```
try {
      // whatever code might throw the exceptions listed below
} catch (LowestException e) {
      // what to do if the code throws a LowestException
} catch (MiddleException e) {
      // what to do if the code throws a MiddleException
} catch (HighestException e) {
      // what to do if the code throws a HighestException
}
```

10. Based on that code, add a new catch block below the one you have for the bad filename
11. In that catch block print "another kind of io exception" and then print the exception
12. Call readFile() from main() (with a valid filename)
13. Run and see what happens. Does it make sense?

Even More Harder Mode

1. Make a copy of the readFile() method called readClosedFile()
2. Move the call to the close() method above where you are reading in the file content
3. Call readClosedFile() from main() with a valid filename
4. Run and see what happens. Does it make sense?

Nightmare Mode

There's one kind of exception we haven't handled: RuntimeException, which is a special subclass of Exception. Unlike other subclasses of Exception, RuntimeException and its subclasses can be thrown by any method, don't have to be (and usually aren't) documented, and you don't have to handle them. Your old friend NullPointerException is a RuntimeException, as is DivideByZero.

1. Make a copy of the readFile() method called readTooFar()
2. If you did this properly, you have will end up with a String variable that points to null
3. Below this, call the length() method on that String variable (put the call inside a print statement)
4. Add a call to readTooFar() from main() with a valid filename
5. Are you sick of exceptions yet? It compiles. Run and see what happens. Does that make sense?
6. Let's fix this; add a new catch block at the end the prior one to catch Exception
7. Inside the catch block, print "something's really messed up now", and print the exception
8. Run it, see what happens

That's about it for exception handling, except that it's possible to pass the exception along further up the call stack. You won't need to do that anytime soon because we're not creating our own exceptions.

Make sure all of your files are added, and then commit them, and push them.