Indiana University
CSCI-C212: Introduction to Software Systems (S25) Lab 12
Joshua Crotts and Muazzam Siddiqui

Exceptions and File I/O

What To Do:

Follow each step carefully. Submit your code to the autograder.

Problem 1

Design the `EchoOdds` class, which reads a file of line-separated integers specified by the user (using standard input), and writes only the odd numbers out to a file of the same name, just with the `.out` extension. If there is a non-number in the file, throw an `InputMismatchException`.

For example, if the user types `"file1a.in"` into the running program, and `file1a.in` contains the following:

```
5
100
25
17
2
4
0
-3848
13
```

then `file1a.out` is generated containing the following:

```
5
25
17
13
```

Another example is, if the user types `"file1b.in"` into the running program, and `file1b.in` contains the following:

```
5
100
25
17
THIS_IS_NOT_AN_INTEGER!
4
0
-3848
13
```

then the program does not output a file because it throws an exception.

Hint: don't be so eager to immediately rush to opening an output stream of some kind! Remember that, if there's a single invalid number in the input file, you do not create a resulting output file. Instead, read the input and if there is an invalid number, immediately throw the aforementioned exception. Otherwise, *then* open the output stream and echo the odd integers. Doing it in this specific order will ensure that an output file is not erroneously created in the event that an invalid number is parsed.

Problem 2

A *stack-based* programming language is one that uses a stack to keep track of variables, values, and the results of expressions. These types of languages have existed for several decades, and in this exercise you will implement such a language.

Design the `StackLanguage` class, whose constructor receives no parameters. The class contains two instance methods: `void readFile(String f)` and `double interpret()`.

- The `readFile` method reads a series of "stack commands" from the file. These can be stored however you feel necessary in the class, but you should not interpret anything in this method, nor should you throw any exceptions. You may want to create a private static class for storing commands.
- The `interpret` method interprets the stored list of instructions. If no instructions have been received by a `readFile` command, throw an `IllegalStateException`. Here are the following possible instructions:

  (a) `DECL` v X declares that v is a variable with value X.

  (b) `PUSH` X pushes a number X to the stack.

  (c) `POP` v pops the top-most number off the stack and stores it in a variable v. If v has not been declared, an `IllegalArgumentException` is thrown.

  (d) `PEEK` v stores the value at the top of the stack in the variable v. If v has not been declared, an `IllegalArgumentException` is thrown.

  (e) `ADD` X adds X to the top-most number on the stack.

  (f) `SUB` X subtracts X from the top-most number on the stack.

  (g) `XCHG` v swaps the value on the top of the stack with the value stored in variable v. If v has not been declared as a variable, an `IllegalArgumentException` is thrown.

  (h) `DUP` duplicates the value at the top of the stack.

  If the command is none of these, then throw an `UnsupportedOperationException`. You may assume that all commands, otherwise, are well-formed (i.e., they contain the correct number of arguments and the types thereof are correct). After interpreting all instructions, the value that is returned is the top-most value on the stack. If there is no such value, throw a `NoSuchElementException`.

  Hint: use a `Map` to store variable identifiers to values.