CMPT 470/816 **HOME EXERCISE 8**

USE OF THE TRY MONAD AND SEQUENCING OPERATIONS

This take home exercise will give you experience in using the Try monad.

Please perform the following within your scala session. The first will import the Try monad. The second will import methods for the StdIn object, by which one can read in text from the terminal.

```
import scala.util._
import scala.io._
```

Please note that, having performed the above, you can perform `StdIn.readLine()` to read in a single line from the standard input as a String.

1) Your first challenge is to read in floating points numbers from the command line, and to return a Try value (which could be either a success or failure, depending on the values entered by the user). Specifically the code will seek to read two such values in succession. If either of the values are not legitimate floating point numbers (that is, if the read line cannot be converted to double values by calling toDouble, in that `StdIn.readLine().toDouble` throws an exception) and immediately (without any further input) returns a Failure. If all three successive values are legitimate such numbers (that is, the read line can successfully be converted to double values by calling toDouble, as in `StdIn.readLine().toDouble`) to add all two together, returning the result as a Try. We provide some hints below:

- To both handle any Double value returned or (alternatively) catch any exceptions triggered, handle please surround the `StdIn.readLine().toDouble` in a creation of a Try, as in `Try(StdIn.readLine().toDouble)`.
- Please remember that given a Try value (just like an Option) value, you can perform a call to *map* on it to process its internal value, and return the resulting value from the expression, itself wrapped in a Try. While the analogy can only be taken so far, you can think of this a bit similar to taking a Vector of a single element and performing a map on that Vector; what is returned is a Vector containing the mapped value. This notion of "Try" (and, in fact, any Monad) as a sort of *container* (just as Vector is a sort of container) will be an important one to which we will return many times for insight.
- Recall that that we have seen previously seen situations where, given a collection (e.g., an Array of lines read from a file) we have sought to map each element of that collection to something that is itself a collection (e.g., an Array of the words that were located within that Array). That is, the function by which you wish to map the first container (and thus which is being passed to the higher order function) returns something that is itself a collection, such as an Array. Recall by order to avoid creating nested collection (a collection within a collection, for our example, an Array within an Array), we used *flatMap*. Similarly, if you wish to map a Try with a function that itself will return a Try, you should use *flatMap*.

Please experiment with the above for both legitimate and illegitimate input.

2) Please convert the above to operate on Int values, and *divide* the first value by the second (again handling exceptions using Try in the same way). Please experiment with both legitimate and illegitimate input, including cases where the second value is 0.

3) Please convert 1) to operate on *three* values, rather than two. Specifically the code will seek to read 3 such values in succession. If any of the three values are not legitimate floating point numbers (that is, if the read line cannot be converted to double values by calling toDouble, in that `StdIn.readLine().toDouble` throws an exception) and immediately (without any further input) returns a Failure. If all three successive values are legitimate such numbers (that is, the read line can successfully be converted to double values by calling toDouble, as in `StdIn.readLine().toDouble`) please add all three together. How would the code change for 4 such values? 5 or more?

Due in class February 14, 2017