CPSC 221
Exception Handling: Format Checker

"A good programmer is someone who always looks both ways before crossing a one-way street."
--Doug Linder

Objectives

- Use Java Exception handling and conditional statements to check the format of a file.

Background

Reading the contents of input files is an important component of many applications. In order to be read and processed correctly, the contents of these files must be in the expected, well-defined format. An incorrectly formatted file could cause a program to crash, but when a well-designed program encounters a problem from which it can recover, it provides an informative message about the problem and continues to run.

Tasks

For this assignment, you will create a program that reads in the contents of a given file and ensures that the data is formatted correctly.

- Your program should be called *FormatChecker.java*.
    - The `FormatChecker` class will evaluate the format of one or more input files, as specified in the command-line arguments.
    - Each file that is in the correct format will be declared VALID.
    - Each file that is not in the correct format will be declared INVALID.
    - All files passed in as command-line arguments should be evaluated and reported. Proper exception handling while parsing each file will allow all files to be evaluated.
    - When the format is invalid, the program should display a short description of why it is invalid.
        - If an exception was thrown while parsing, the short message should be the output of the exception's toString() method - do not print a stack trace for any exceptions found

like `FileNotFoundException`, `NumberFormatException`, and `InputMismatchException`.
        - If the formatting problem was discovered by a condition check, you will need to come up with an appropriate short description of the problem.

- The *valid* format rules are as follows:
    - The first row contains two white-space-separated, positive integers.
        - The first integer specifies the number of *rows* in a grid.
        - The second integer specifies the number of *columns* in the grid.
    - Each subsequent row represents one row of the grid and should contain exactly one white-space-separated double value for each grid column.
    - This is an example of a correctly formatted file:

      5 6
      2.5 0 1 0 0 0
      0 -1 4 0 0 0
      0 0 0 0 1.1 0
      0 2 0 0 5 0
      0 -3.14 0 0 0 0

- Any one of the following issues will make the file format *invalid*:
    - The file can't be found.
    - The file contains data in the wrong format.
        - The values don't match the expected data types.
        - There are fewer rows and/or columns of data than specified.
        - There are more rows and/or columns of data than specified.

This zip file includes both valid and invalid input files that can be used for testing.

Running the Program

Your program should be run from the command-line using the following format:

```
$ java FormatChecker file1 [file2 ... fileN]
```

where the names of one or more input files are given.

For instance, if you run the program with only one file called *valid1.dat*, this is the command you should use:

```
$ java FormatChecker valid1.dat
```

and this should be your output:

```
valid1.dat
VALID
```

If you run the program with four files named *valid1.dat*, *valid2.dat*, *invalid1.dat*, and *valid3.dat*, here are command-line arguments you would use and the expected output:

```
$ java FormatChecker valid1.dat valid2.dat invalid1.dat valid3.dat
valid1.dat
VALID

valid2.dat
VALID

invalid1.dat
java.lang.NumberFormatException: For input string: "X"
INVALID

valid3.dat
VALID
```

If you run the program with two files named *noSuchFile* and *valid1.dat*, here are command-line arguments you would use and the expected output:

```
$ java FormatChecker noSuchFile valid1.dat
noSuchFile
```

java.io.FileNotFoundException: noSuchFile (The system cannot find the file specified)
INVALID

valid1.dat
VALID

Because arguments are required, you should also check for at least one argument and display a helpful usage message if no arguments are given

$ java FormatChecker
Usage: $ java FormatChecker file1 [file2 ... fileN]

Because there are a variety of ways of parsing file content and detecting format problems, your messages and reported exceptions may not *exactly* match the examples, above, but the style of your output should match what is shown.

Documentation: README

Update the plain text file called README and write your name and class section on the top.

1. Fill out the **Overview** section
   Concisely explain what the program does. If this exceeds a couple of sentences, you are going too far. I do not want you to just cut and paste, but paraphrase what is stated in the project specification.
2. Fill out the **Compiling and Using** section
   This section should tell the user how to compile and run your code. It is also appropriate to instruct the user how to use your code. Does your program require user input? If so, what does your user need to know about it to use it as quickly as possible?
3. Fill out the **Discussion** section
   Think about and answer the following reflection questions as completely as you can. You may not have an "earth-shattering" reflection response to each one but you should be as thoughtful as you can.
   Reflections...
   - What problems did you have? What went well?
   - What process did you go through to create the program?
   - What did you have to research and learn that was new?
   - What kinds of errors did you get? How did you fix them?
   - What parts of the project did you find challenging?

- Is there anything that finally "clicked" for you in the process of working on this code?
- Is there anything that you would change about the lab?
- Can you apply what you learned in this lab to future projects?

4. Fill out the **Testing** section
   You are expected to test your projects before submitting them for final grading. Pretend that your instructor is your manager or customer at work. How did you ensure that you are delivering a working solution to the requirements?

Grading

Points will be awarded according to the following breakdown:

Tasks	Points
Functionality - Correctly identifies valid/invalid files	15
Quality of Code – README, comments, formatting, conventions, meaningful naming, etc.	5

Required Files

Submit copies of the following files:

- FormatChecker.java
- README.txt

You should test your program thoroughly before submitting it. Your program should run from the command line and should provide outputs to the console based on the given command line options.