Assignment 4

Input Validation Methods	10 pts.
Pattern Display Method	15 pts.
Random Numbers in Files Method	10 pts.
File Analysis	15 pts.

TOTAL: 50 pts.

General Requirements

- *Add comments to the source code you are writing:*
  - *Describe the purpose of every variable*
  - *Explain the algorithm you are using for solution*
  - *Add proper documentation comments for all methods. Include @param, @return, and @throws tags*
  - ***Generate Javadoc files for each of your projects**.*
- ***Please see code example MethodsAndExceptions.java for***
  - ***sample exception throwing/catching code***
  - ***how to organize your test cases***
  - ***how to document your code***
- *What to turn in:*
  - *The entire project*
  - ***ATTENTION:** Javadoc files must be generated in your project*

Input Validation Methods

Convert the code you used in Asg. 3 for user input validation into two methods:
`int validateInt(String prompt)` and
`double vlidateDouble(String prompt)`
Each of these methods does the following

- takes prompt string as a parameter,
- asks user for input using the prompt sting provided,
- validates input, making sure the type of the numeric value provided by the user is correct.
  The methods handle exceptions as they arise, use exceptions to determine if the type of the number entered was correct. The input prompt is repeated in the loop until the user provides the numeric value of correct type.
- returns the integer or double value (according to the name of the method) to the calling code.

From now on you can call these methods whenever user input validation is needed.
Please see file **InputValidationMethods.java** – it contains a good staring platform for your program.

Pattern Display Method

Write a method
`public static void patternDisplay(int size, char c, boolean direction) throws IllegalArgumentException` that prints patterns of symbols.
Parameter `size` defines the size of pattern, in particular the size of the longest sequence of characters in the pattern.
Parameter `c` defines the choice of character that is used to display the patters
Parameter `direction` defines the direction of the pattern.
Please see two function calls and the patterns they produced below.
The method must throw `IllegalArgumentException` when the value of parameter `size` is a negative number.

Test your new method in main(). Do not ask user for input, instead hard-code all the values you will be using to test your code. Make sure to use try/catch block in order to test that the exception is being thrown by the method when the argument size is negative. Make sure the method works well for correct arguments too.

```
patternDisplay(7, '+', true)
+
++
+++
++++
+++++
++++++
+++++++


patternDisplay(4, '#', false)

####
###
##
#
```

Random Numbers in Files Method

Name this project RandomNumsInFiles.java

This problem is based on the "File with Random Numbers" problem from Assignment 3.

Write method `void randomNumsFile(String fileName, int howMany, int rangeFrom, int rangeTo) throws IOException` that takes file name as a string and 3 integer as parameters. The method generates random integers and saves them in a file, placing each new number on a new line. Parameters represent the following

1. `String fileName` - file name. File created if it does not exist. If the file does exist its content is being replaced with this method.
2. `int howMany` - Number of random numbers to be generated
3. `int rangeFrom` and `int rangeTo` – The range of random numbers to be generated. If the `rangeFrom` is smaller than `rangeTo` - swap them.

This method `throws IOException` when the file fails to open for writing.

Test your method by calling it in main(). Make sure to use try-catch block to handle IOExcptions your new method may throw. Main() should not throw exceptions.
Do not use user input for testing, **create test cases** that are easy to follow when reading your code.

File Analysis Method

Write method `int fileAnalysis (String inputFileName, String outputFileName)` that analyzes the sequence of integers stored in a file with a given name (`inputFileName`), and calculates the following statistics about the data stored in file:

- the number of integers stored,
- the largest and the smallest number stored in file,
- the total of all numbers stored in the file.

All that statistics must be stored in another file. The name of that output file is provided in parameter `outputFileName`.
The method returns the number of the integers processed.
Assume that all the numbers stored in the input file are integers and stored one number per line.
The method `throws IOException` when the files fail to open for reading and writing. If the input file does not exist, the method returns 0.

Sample statistics output file format is the following:

```
Numeric data file name: nums.txt
Number of integers: 5
The total of all integers in file: 15
The largest integer in the set: 5
```

```
The smallest integer in the set: 1
```

Test new method in main(). Make sure to use try/catch block to handle `IOExceptions` it may throw. Do not use user input for testing, **create test cases** that are easy to follow when reading your code.

Suggested testing strategy: create a small testing text file containing 3 – 5 numbers that are easy to add up and where smallest and largest numbers are obvious. Make sure your method runs well with this data set. Do not use user input in your test cases.
Sample file contents:
5
3
1
4
2

Requirements:

1. DO NOT use arrays or ArrayLists to store the numbers in the integer set. Numbers must be analyzed on the spot. Use the algorithm below (from Asg. 3) to find the smallest and the largest number in the set. Never assume that all the numbers in the set are positive.
2. For the analysis go through the set of numbers in the file ONCE.
3. main() cannot throw any exceptions. Please handle exceptions in main() properly.

"The greatest of a set of numbers" algorithm:

1. *Get (input) the first number, set it to be the greatest for now.*
2. *Get the next number.*
3. *If the new number is bigger than the "greatest" one – set the new one to be the "greatest".*
4. *Repeat steps 2 - 4 until all the numbers are consumed.*