CPSC 233 – Coding Challenge 5 – Practice 1

This is a practice coding challenge. You may ask other students and your TA any questions you wish. Note that you are expected to complete the actual coding challenge independently. Having a solution for this coding challenge will not help you come up with your own solution for the actual coding challenge. Only if you can solve this practice independently can you be confident that you can complete the actual coding challenge successfully. See previous coding challenges for rules for coding challenges and best practices for success.

Remember to always create a 'skeleton' first and compile and test this using the provided JUnit test. For this coding challenge, you will be asked to create one class. Make sure you create a skeleton of all classes first. You can upload your solution to WebCAT directly. You do not need to place it in a zip file.

Before a method ends, make sure to close any file that you have opened. Not doing so may cause tests to fail. If using Scanner, make sure that a method only has a single instance of Scanner open at a time. (Not doing so may also cause problems with the tests.)

Create a class called FileExercises that contains the following methods.

- public int checkNumber(int num)
  - Return the number passed in as an argument if this number is greater than zero.
  - Throw an IOException otherwise.

- public void append(String text, String filename)
  - Append text to the text file filename.
  - If any errors occur while opening the file or writing to the file, do nothing at all.

- public void getLetters (String inputFileName, String outputFileName)
  - Take the first letter from each word in the input file (a text file), translate them to upper case, concatenate them together, and place the result in the output file (also a text file).
  - If the input file does not exist or the output file can't be created, your method should throw a FileNotFoundException.

- public int count(String word, String filename)
  - Counts the number of times word appears in the text file.
  - You can assume there is no punctuation in the file.
  - You may assume that all white space is either a new line or a single space.
  - Do not include the times that word appears as a substring of another word in the file.
  - If an exception occurs, return -1.

- public double sumNumbers(String inputFilename)
  - inputFilename is the name of a binary file.
  - The first entry is of type int, indicating the amount of double numbers in the file.
  - Read all the doubles, sum them and return the result.
  - If an exception occurs, return -1.0.