**Assignment 2**

**Due date**

* 11:59:59 PM EST, Feb 23rd

**Git url**

* <https://classroom.github.com/a/hyyvHRmy>

Submit your code as per the provided instructions.

**Assignment Goal**

Apply the Observer Pattern and the design principles you have learnt so far to develop a program for the given problem.

**Team Work**

* No team work is allowed. Work individually. You cannot discuss the assignment with ANYONE other than the instructor and TA.

**Programming Language**

You are required to program using Java.

**Compiling and Running Commands**

* Follow instructions in the README file for compilation and running.

**Policy on sharing of code**

EVERY line of code that you submit in this assignment should be written by you. Do NOT show your code to any other student. Do not copy any code from any online source. Code for File I/O and/or String operations, if found online, should be clearly cited, and you cannot use more than 5 lines of such online code.

Code downloaded in its entirety from an online repository of code (GitHub, BitBucket, etc.) and submitted as student's own work, even if citied, is considered plagiarism.

Code snippets, for File I/O, if used from an online source should be cited by mentioning it in the README and also in the documentation of every source file in which that code appears.

Post to the piazza if you have any questions about the requirements. Do NOT post your code to the piazza asking for help with debugging.

**Project Description**

Assignment Goal: Observe a stream of numbers and calculate different metrics.

* The program accepts one input file. This input file contains integer and/or floating point numbers.
* Each number read from the input file is passed to a *NumberProcessor*.
* The *NumberProcessor* operates as indicated below, for each number received.
  + An event is triggered based on the data type of the number. Below are the possible events that can be triggered.
    - **INTEGER\_EVENT** - Triggered if the number read is an integer.
    - **FLOATING\_POINT\_EVENT** - Triggered if the number read is a floating point number (float or double).
  + Based on the type of event triggered, the appropriate *Observers* are notified and the number is provided to them along with the notification.
  + Each *Observer* then carries out some operations using the provided number and stores the result of the operation.
* Once all the numbers have been read in from the file, the *NumberProcessor* would trigger a **PROCESSING\_COMPLETE** event and would notify all the observers. The observers, when notified of this event, should persist the results into the corresponding output files.

**Observers**

For this assignment, you are to write three observers. Below are the observers and the their respective responsibilities.

* **RunningAverageObserver**
  + Running average is basically the average of the most recent W numbers, where W is the size of the window (a fixed size FIFO queue). See below example for better understanding.
    - Let 1,2,3,4,5,6,7 be the list of numbers to be processed.
    - Let the window size = 3.
    - Number processed = 1. Running average of [1] = 1.0
    - Number processed = 2. Running average average of [1,2] = 1.5
    - Number processed = 3. Running average of [1,2,3] = 2.0
    - Number processed = 4. Running average of [2,3,4] = 3.0
    - Number processed = 5. Running average of [3,4,5] = 4.0
    - Number processed = 6. Running average of [4,5,6] = 5.0
    - Number processed = 7. Running average of [5,6,7] = 6.0
  + The RunningAverageObserver is very particular about not losing precision and listens to INTEGER\_EVENT updates. This is because addition of floating point numbers can result in loss of precision.
  + Each time this observer is notified, the new number read from the input file is provided.
  + The running average is then computed using the new number.
  + The size of the window for running average calculations is provided via the command-line argument -DrunAvgWindowSize (see README for usage information).
  + The calculated running average is then stored in the results.
  + The path of the file into which the results need to be persisted is provided via command-line argument -DrunAvgOutFile (see README for usage information).
  + *Note: running averages can be rounded to two decimal places.*
  + **Challenge: Can you make each running average calculation a constant time operation?**
* **TopKNumbersObserver**
  + Top K numbers are the largest K numbers among all the numbers that have processed so far. See below example for better understanding.
    - Let 1, 2, 3.45, 4.6, 4.65, 6.8, 7 be the list of numbers to be processed.
    - Let k = 3.
    - Number processed = 1. Top k = [1.0]
    - Number processed = 2. Top k = [2.0, 1.0]
    - Number processed = 3.45. Top k = [3.45, 2.0, 1.0]
    - Number processed = 4.6. Top k = [4.6, 3.45, 2.0]
    - Number processed = 4.65. Top k = [4.65, 4.6, 3.45]
    - Number processed = 6.8. Top k = [6.8, 4.65, 4.6]
    - Number processed = 7. Top k = [7, 6.8, 4.65]
  + The TopKNumbersObserver does not mind floating point numbers and therefore listens to both INTEGER\_EVENT and FLOATING\_POINT\_EVENT.
  + Tip - Use [BigDecimal](https://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html" \l "compareTo(java.math.BigDecimal)) to compare the numbers as either floating points or doubles.
  + **Question: What data structure would you use to compute the top k numbers?**
  + Each time this observer is notified, the new number read from the input file is provided.
  + The top K numbers are then determined.
  + The value of K is provided via command-line argument -Dk (see README for usage information).
  + The top K numbers are then stored in the results.
  + The path of the file into which the results need to be persisted is provided via command-line argument -DtopKNumOutFile (see README for usage information).
  + *Note: Numbers can be rounded to two decimal places before comparing.*
* **NumberPeaksObserver**
  + A number peak is the number just preceding a dip in the value of numbers being processed. See below example for better understanding.
    - Let 1, 2, 1.8, 2.2, 3, 3.45, 4.65, 4.6, 6.8, 7 be the list of numbers to be processed.
    - Number processed = 1. No peak yet as this is the first number.
    - Number processed = 2. No peak yet as 2 > 1.
    - Number processed = 1.8. Found a peak as 1.8 < 2. The peak is 2.
    - Number processed = 2.2. No peak yet as 2.2 > 1.8.
    - Number processed = 3. No peak yet as 3 > 2.2.
    - Number processed = 3.45. No peak yet as 3.45 > 3.
    - Number processed = 4.65. No peak yet as 4.65 > 3.45.
    - Number processed = 4.6. Found a peak as 4.6 < 4.65. The peak is 4.65
    - Number processed = 6.8. No peak yet as 6.8 > 4.6.
    - Number processed = 7. No peak yet as 7 > 6.8.
  + The NumberPeaksObserver does not mind floating point numbers and therefore listens to both INTEGER\_EVENT and FLOATING\_POINT\_EVENT.
  + Tip - Use [BigDecimal](https://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html" \l "compareTo(java.math.BigDecimal)) to compare the numbers as either floating points or doubles.
  + Each time this observer is notified, the new number read from the input file is provided.
  + Whether a peak was encountered or not is then determined.
  + If a peak was found, then the value of that peak is stored in results.
  + The path of the file into which the results need to be written is provided via command-line argument -DnumPeaksOutFile (see README for usage information).
  + *Note: Numbers can be rounded to two decimal places before comparing.*
  + *Note: Only peaks need to be stored into results.*
* *Note: All the observers should also be registered for PROCESSING\_COMPLETE updates. Otherwise, they will not know when to persist the results to the output files.*

The following rules MUST be followed.

1. The program should not read in all the numbers, store it in a data structure, and then process them.
2. An observer should be notified only when the corresponding event has been triggered. Use filters for this purpose.

**INPUT FORMAT**

Your program should accept 6 arguments from the commandline. See README on Github for more information on these.

1. *-DinputNumStream* - Input file path (string).
2. *-DrunAvgWindowSize* - Size of the window for running average calculations (integer). Must be > 0.
3. *-DrunAvgOutFile* - Name of the output file to which running averages are written (string).
4. *-Dk* - Max size of the list containing the top K numbers (integer). Must be > 0.
5. *-DtopKNumOutFile* - Name of the output file to which the top K numbers are written (string).
6. *-DnumPeaksOutFile* - Name of the output file to which the peaks in the number stream are written (string).

*Note: Input file will be well formatted.*

**EXAMPLES**

Sample run command:

ant -buildfile numberPlay/src/build.xml run \

-DinputNumStream="input.txt" \

-DrunAvgWindowSize=3 \

-DrunAvgOutFile="run-avg-out.txt" \

-Dk=3 \

-DtopKNumOutFile="top-k-out.txt" \

-DnumPeaksOutFile="peaks-out.txt"

|  |  |  |  |
| --- | --- | --- | --- |
| **input.txt** | **run-avg-out.txt** | **top-k-out.txt** | **peaks-out.txt** |
| 1  2  1.8  2  2.2  3  3.45  4  4.65  4.6  5  6  6.8  7 | 1  1.5  1.67  2.33  3  4  5  6 | [1]  [2, 1]  [2, 1.8, 1]  [2, 2, 1.8]  [2.2, 2, 2]  [3, 2.2, 2]  [3.45, 3, 2.2]  [4, 3.45, 3]  [4.65, 4, 3.45]  [4.65, 4.6, 4]  [5, 4.65, 4.6]  [6, 5, 4.65]  [6.8, 6, 5]  [7, 6.8, 6] | 2  4.65 |

**NOTES ON GRADING**

* Class participation points will be given to students who piazza with good questions seeking clarification on the assignment, and also to those who respond and participate to help make the discussion interesting and informative.

**Compiling and Running Java code**

* Your submission must include a readme in markdown format with the name**README.md**.
* Your README.md file should have the following information:
  + instructions on how to compile the code
  + instructions on how to run the code
  + justification for the choice of data structures (in terms of time and/or space complexity).
  + justification for the design patterns, other than Observer, that have been incorporated in this assignment.
  + citations for external material utilized.
* You should have the following directory structure (username would be substituted with your github username).
* ./csx42-spring-2020-assign2-username
* ./csx42-spring-2020-assign2-username/numberPlay
* ./csx42-spring-2020-assign2-username/numberPlay/src
* ./csx42-spring-2020-assign2-username/numberPlay/src/numberPlay
* ./csx42-spring-2020-assign2-username/numberPlay/src/numberPlay/subject
* ./csx42-spring-2020-assign2-username/numberPlay/src/numberPlay/subject/SubjectI.java
* ./csx42-spring-2020-assign2-username/numberPlay/src/numberPlay/observers
* ./csx42-spring-2020-assign2-username/numberPlay/src/numberPlay/observers/ObserverI.java
* ./csx42-spring-2020-assign2-username/numberPlay/src/numberPlay/util
* ./csx42-spring-2020-assign2-username/numberPlay/src/numberPlay/util/FileProcessor.java
* ./csx42-spring-2020-assign2-username/numberPlay/src/numberPlay/util/PersisterI.java
* ./csx42-spring-2020-assign2-username/numberPlay/src/numberPlay/util/RunningAverageResultsI.java
* ./csx42-spring-2020-assign2-username/numberPlay/src/numberPlay/util/TopKNumbersResultsI.java
* ./csx42-spring-2020-assign2-username/numberPlay/src/numberPlay/util/NumberPeaksResultsI.java
* ./csx42-spring-2020-assign2-username/numberPlay/src/numberPlay/util/RunningAverageData.java
* ./csx42-spring-2020-assign2-username/numberPlay/src/numberPlay/util/TopKNumbersData.java
* ./csx42-spring-2020-assign2-username/numberPlay/src/numberPlay/util/NumberPeaksData.java
* ./csx42-spring-2020-assign2-username/numberPlay/src/numberPlay/driver
* ./csx42-spring-2020-assign2-username/numberPlay/src/numberPlay/driver/Driver.java
* ./csx42-spring-2020-assign2-username/numberPlay/src/build.xml
* ./csx42-spring-2020-assign2-username/.gitignore
* ./csx42-spring-2020-assign2-username/README.md
* [Other Java files you may need]

**Code Organization**

* Your directory structure should be EXACTLY as given in the code template
  + Use the command on linux/unix to create an archive: *tar -cvzf csx42-spring-2020-assign2-username.tar.gz csx42-spring-2020-assign2-username/*.
  + Use the command on linux/unix to extract the file: *tar -zxvf csx42-spring-2020-assign2-username.tar.gz*.

**Submission**

* Make sure all class files, object files (.o files), executables, and backup files are deleted before creating a zip or tarball (update .gitignore file if need be). To create a tarball, you need to "tar" and then "gzip" your top level directory. Create a tarball of the directory csx42-spring-2020-assign2-username. We should be able to compile and execute your code using the commands listed above.
* Instructions to create a tarball
  + Make sure you are one level above the directory csx42-spring-2020-assign2-username.
  + tar -cvzf csx42-spring-2020-assign2-username.tar csx42-spring-2020-assign2-username/
  + gzip csx42-spring-2020-assign2-username.tar
* Upload your assignment to Blackboard, assignment-2.

**General Requirements**

* Start early and avoid panic during the last couple of days.
* Separate out code appropriately into methods, one for each purpose.
* You should document your code. The comments should not exceed 72 columns in width. Use javadoc style comments if you are coding in Java. Include javadoc style documentation.
  1. Comments for classes should include (1) purpose the class servers, (2) name of the author (you).
  2. Comments for instance/static variables should include use of the variable.
  3. Comments for methods should describe (1) purpose of the method, (2) each argument the method expects, and (3) return valueand/or exceptions thrown.
* Do not use "import XYZ.\*" in your code. Instead, import each required type individually.
* All objects, in Java, that may be needed for debugging purposes should have the "toString()" method defined. By default, just place a toString() in every class.
* Every class that has data members, should have the following.
  1. Corresponding accessors and mutators (unless the data member(s) is/are for use just within the method).
  2. Overriden equals() and hashCode() methods.
  3. Overriden toString() method.

**Design Principles to be followed**

1. Programming to the interface.
2. Separation of concerns.
   * Modularize your code such that each method and/or class has a well-defined responsibilities.
   * Move out repeated operations into separate methods to promote code reuse.
3. Use the validator design explained in class for command-line and input validation.
4. Semantically correct variable/method/class names.
5. Adhere to java naming conventions.
6. The Subject holds the observers in a collection or an array.
7. Observers should not hold reference to the Subject.
8. Observers should not directly or indirectly call methods on the Subject.
9. Generic exceptions should not be thrown or caught (*throw new Exception(...)* or *catch (Exception e)*).
10. Program should not abruptly terminate. Meaningful error messages should be displayed and the program should gracefully shutdown.

**Late Submissions**

* The policy for late submissions is that you will lose 10% of the grade for each day that your submission is delayed. **There is NO difference in penalty for assignments that are submitted 1 second late or 23 hours late**.

**Grading Guidelines**

Grading guidelines have been posted[here](https://sites.google.com/binghamton.edu/csx42/spring-2020/assignments/assign2/grading-instructions-html).