Assignment 3

Due date

- 11:59 PM EST, Mar ~~24th~~ 29th, 2020.

Github link

The git links are the following:

- https://classroom.github.com/g/8Dn2RMJS

Submit your code as per the provided instructions.

Assignment Goal

- Application of design principles for a simple multi-threaded application, using wait() and notifyAll(). Socket programming.
- **Students can work in teams of up to 2 members.**

Programming Language

You are required to program using Java.

Compilation Method

- You are required to use ANT for compilation of code written in Java.
- Your code should compile and run on *remote machines* or the *debian-pods* in the Computer Science lab in the Engineering Building.

Policy on sharing of code

- EVERY line of code that you submit in this assignment should be written by you or be part of the code template provided for this assignment. Do NOT show your code to any other student. Our code-comparison software can very easily detect similarities.

Project Description

- The following two programs should be run.
    1. **PrimeDetector** Use multiple threads to,
        - Poll for integers from a file.

- check if the number is prime.
- If the number is determined to be prime, then store it in a shared results instance.
- Each time a prime is stored, the results instance should send the prime to a PersisterService that is listening on a specified IP address and port number (Use socket programming for this).
- Each prime received by the PersisterService is persisted to the specified output file.
- From the command line, accept the following as input:
    - *inputFile*: The name of the input file.
    - *numThreads*: The number of threads to be used: referred to as NUM_THREADS below.
    - *capacity*: Capacity of the results data structure.
    - *persisterServiceIp*: IP Address of the PersisterService (Note that the PersisterService should be started before PrimeDetector).
    - *persisterServicePort*: Port number on which the PersisterService is listening for data.
    - *debugValue*: an integer that controls what is printed on stdout.
        - Validate that the correct number of command line argumets have been passed.
        - Validate that the value of NUM_THREADS is between 1 and 5.
        - Validate that the DEBUG_VALUE is between 0 and 4.
- The Driver code should create an instance of CreateWorkers and pass an instance of the FileProcessor, Results, and IsPrime to the constructor of CreateWorkers. The Driver should invoke the method startWorkers(...) in CreateWorkers and pass the NUM_THREADS as an argument.
- CreateWorkers' startWorkers(...) method should borrow NUM_THREADS threads (via the threaded class WorkerThread), start them and join on them. The instances fo FileProcessor, Results, and IsPrime should be passed to he constructor of the worker thread class.
- Design a thread pool to borrow threads (it is ok in this assignment to not return threads to the pool). Implement the "borrow" method in the thread pool, and then any other method you need.
- The *run* method of the worker thread should do the following till the end of file is reached:
    - While the end of file has not been reached:
        - Invoke a method in the FileProcessor to retrieve one line, as string from the input file.
        - Check if it is a prime number using the IsPrime utility.
        - Store the prime number in a data structure in the Results instance. If the Capacity of the internal data structure in results is reached, then the thread waits. Otherwise, it should *notifyAll*.
        - Once the end of the file is reached, the worker thread stores the "STOP" message in the results instance and terminates.

        - The choice of data structure used in the Results class should be justified in the README.txt in terms of space and/or time complexity.
        - The Results class should run a thread (let's call this DataSender) that constantly sends data to the PersisterService, using sockets (Use socket programming for this). So, the DataSender thread would be the client and the PersisterService would be the server. If the internal data structure in results is empty, then DataSender thread should wait. Otherwise, it should *notifyAll*.
    2. **PersisterService**
        - The following command line arguments should be accepted.
            - *port*: Port number on which the server should listen.
            - *outputFile*: Name of the output file to which the data received on the port should be written.
        - The persister should constantly listen on the specified port number.
        - Any data received should be written to the output file.
        - If the "STOP" message is received, then the PersisterService should persist any unpersisted data, close the output file, and terminate.
        - The PersisterService should be started before the PrimeDetector.
- Assume that the input file will have one string per line and no white spaces. Also assume that the input strings in the file, if a number, can be converted to integers.
- Except in the Logger class and the thread pool pattern (if you plan to use it) do not make any other method static. Here is code you can extend and use for MyLogger.java
- The DEBUG_VALUE, read from the command line, should be set in the Logger class. Use the DEBUG_VALUE in the following way:
    - DEBUG_VALUE=4 [Print to stdout everytime a constructor is called]
    - DEBUG_VALUE=3 [Print to stdout everytime a thread's run() method is called]
    - DEBUG_VALUE=2 [Print to stdout everytime an entry is added to the Results data structure]
    - DEBUG_VALUE=1 [Print to stdout the contents of the data structure in the ~~store~~ Results instance]
    - DEBUG_VALUE=0 [No output should be printed from the application, except the line "The sume of all the prime numbers is: XYZ"]
- The Logger class should have a static method to writeMessage(...).
- Place the FileProcessor.java in the util/ folder.

Code Organization and Command-Line Arguments

- Follow the directory structure of the previous assignments.
- To allow for running multiple programs, add the following command-line arguments (apart from the ones mentioned above) to the build.xml file.
    1. *primeDetector*: This should run the PrimeDetector program with the rest of the command-line args.
    2. *persisterService*: This should run the PersisterService program with the rest of the command-line args.
- For example, to run the PersisterService, the below command would be used.

- *ant -buildfile src/build.xml run -DpersisterService -Dport=9090 -DoutputFile=output.txt*

Submission

- Make sure all class files, object (.o files), executables, and backup files are deleted before creating a zip or tarball. To create a tarball, you need to "tar" and then "gzip" your top level directory. Create a tarball of the directory csx42-spring-2020-assign3-XXX. We should be able to compile and execute your code using the commands listed below.
  - *tar -cvzf csx42-spring-2020-assign3-XXX.tar.gz csx42-spring-2020-assign3-XXX/.*
  - To extract files from the tarball, use the command *tar -xvzf csx42-spring-2020-assign3-XXX.tar.gz* .
- Upload your assignment to Blackboard, assignment-3.

General Requirements

- Start early and avoid panic during the last couple of days.
- Submit a README.md file (placed at the top level). The README.txt file should be filled out as described in that file.
- Apply all design principles (wherever applicable).
- Separate out code appropriately into methods, one for each purpose.
- You should document your code. The comments should not exceed 72 coloums in width. Use javadoc style comments if you are coding in Java.
- 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 corresponding accessors and mutators (unless the data member(s) is/are for use just within the method.).
- If a class does not implement an interface, you should have a good justification for it. For example, it is ok to have an abstract base class and a derived class, wherein both do not implement interfaces. Note that the Driver code is written by end-users, and so the Results class must implement the interface, or else the source code for Results will have to be exposed to the end-user.
- Include javadoc style documentation for at least 3 classes. It is acceptable for this assignment to just have one parameter of a method described for each method's documentation.
- For the classes provided in the code template, add interfaces as appropriate

Design Requirements

- Prevent race conditions by appropriately using wait() and notifyAll().
- Input file should be processed one line at a time.
- Worker threads should not read in all the lines at once, store it and then process them.

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.