Assignment 2

Due date

- Due by 11.59 PM EST on Sept ~~29th~~ 30th.

Submit your code as per the provided instructions. A signup sheet will be provided to you during class to setup an appointment with the TA to provide a demo of your project.

Updates

Assignment Goal

Application of design principles for a simple multi-threaded application.

Team Work

- CS 542: team work is NOT allowed. Work individually.
- CS 442: teams of two students.

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.cs.binghamton.edu* 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 group. Our code-comparison software can very easily detect similarities.
- Post to the listserv if you have any questions about the requirements. Do NOT post your code to the listserv asking for help with debugging.

Project Description

- From the command line, accept the following as input:
  - The name of the input file (referred to as input.txt below)
  - The name of the output file (referred to as output.txt below)
  - The number of threads to be used: referred to as NUM_THREADS below
  - Debug Value: 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 3.
    - Validate that the DEBUG_VALUE is between 0 and 4.

- Update from Assignment-1: There are 7 courses (A, B, C, D, E, F, G) being offered in the summer session. The capacity for each course is 60. The total number of students is 80. Each student is required to register for 5 courses. The student is asked to provide a preference for each of the courses. Top preference is specified as "1", while the lowest preference is specified as "7".
- The Driver code should create an instance of CreateWorkers and pass an instance of the FileProcessor, Results, other instances needed by the Worker 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 create NUM_THREADS threads (via the threaded class WorkerThread), start them and join on them. The instances fo FileProcessor, Results, and classes needed for the scheduling should be passed to he constructor of the worker thread class.
- 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 read one line as a string
    - Run your algorithm to assign courses to this student.
    - Store the results in the data structure in the Results instance
- 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 implement an interface, StdoutDisplayInterface. This interface should have a method writeSchedulesToScreen(...). The driver should invoke this method on the Results instance to print the schedules, average preference score, etc. (similar format as Assignment-1) for all the students.
- The Results class should implement an interface, FileDisplayInterface. This interface should have a method writeSchedulesToFile(...). The driver should invoke this method on the Results instance to print the schedules, average preference score, etc. (similar format as Assignment-1) for all the students to the output file.
- Use an Object Pool instance to manage the 7 courses. If your algorithm requires you to return a spot in a course, then you should use the ObjectPool API for this purpose. You should ObjectPool API to request a spot in a course, check availability, etc.
- Assume that the input file will have one unique string per line, no white spaces, and no empty lines. Also assume that the intput strings in the file do not have errors. The entries in every line are space delimited.
- Except in the Logger, Object Pool class, do not make any other method static.
- 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]
  - DEBUG_VALUE=0 [No output should be printed from the application, except the line "The average preference value is X.Y"]
- The Logger class should have a static method to ~~writeOuput(...).~~ writeMessage(...).
- Place the FileProcessor.java in the util/ folder.

Code Organization

- Your directory structure should be exactly as given in the code template
  - Download the ANT based tarball here. Use the command on linux/unix: *tar -xvf firstName_lastName_assign2.tar.gz*.
  - Her is the script that you can use to run the project, in case you do not want to use ANT to run it. Note that you still have to use ANT to compile the code.
    - Place the run.sh script at the same level as the BUILD folder
    - In the run.sh script, the target has been named as "driver". You can change it, if you like.

    - Use it in the following way: ./run.sh driver input.txt output.txt 4 3
    -
- If you are working on a team project, change the top level directory to firstName_lastName_firstName_lastName_assign2

- You can compile and run the code with the commands listed in README.txt file, which you can find once you untar the provided code template.

Submission

- Read this file for general guidelines on how to prepare a README for your submission.
- Run "ant clean" and make sure all class files, object files (.o files), executables, and backup files are deleted before creating a zip or tarball. To create a tarball, use the provided (or your own) target in build.xml. The command to "untar" should be specified in the README.txt.
- Both the team members should submit.

General Requirements

- Start early and avoid panic during the last couple of days.
- Submit a README.txt 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.
- 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. It is acceptable for this assignment to just have the return type described for each method's documentation.
- For the classes provided in the code template, add interfaces as appropriate

Design Requirements

Late Submissions

- Same as Assignment-1.

Grading Guidelines

Grading guidelines have been posted here.

mgovinda at cs dot binghamton dot edu
Back to CSX42: Programming Design Patterns