Assignment 1

Due date

- CS 542: Due by 11.59 PM EST on February 23rd.
- CS 442: Due by 11.59 PM EST on February 25th.

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

- Mon Feb 10 14:58:59 EST 2014: Assignment posted
- Wed Feb 12 21:12:47 EST 2014: Search file terms do not contain the course numbers.
- Perl script (Thanks, Hemanth Rao)
- Shell script (Thanks, Niranjan Bhalerao)

Assignment Goal

Application of design patterns/principles for a simple multi-threaded application. All design patterns discussed till the assignment due date may be applicable.

Team Work

CS 542: You are required to work alone on this project.
CS 442: You have the option of working in teams of 2 students each.

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 *bingsuns* 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

Student Registration Log

- The functionality listed below should be placed in appropriate classes/methods.
- Design a data structure (or use a built-in one), for RegistrationStore class, that stores StudentInfo records. Each StudentInfo record has the following: student first name, student last name, instructor's first name, and course number.
    - first name, last name, instructor's first name: String
    - course number: integer
- The RegistrationStore should have a displayData(...) method that should print all the data it has to stdout.
- Spawn NN threads (use PopulateWorker class to spawn the threads) to read data from a file, dataFile, to populate the RegistrationStore. The file should have at least 100 entries. All the NN threads should read from the same file. No entry should be read more than once, and all entries should be read.
- The format of the dataFile should be the following

  ```
  John Doe Horowitz 240
  Mary Jane Gates 442
  ```

  The above format is to easy to generate with a simple script/program. Ofcourse, you can use more accurate course details if you like. If you write a program to generate these entries, the code for that should be placed in the *util* package.
- Every entry from dataFile should be considered unique (i.e. generate the data so that all strings are unique).
- Spawn MM threads (use SearchWorker class to spawn the threads) that read from another file, searchFile, that has a list of words. For each word, search in the RegistrationStore if that word exactly matches any data in the store. NEW: Assume that the search file only has entries for the strings, and not the integer course number. The entire entry (first name, last name, instructor name, and course number) for each hit should be stored in a separate data structure (this data structure should be in the Results.java class).
- Each search word should be read only once from the searchFile. The search word can be from any of the following: first name, last name, instructor's first name. Note that your design of the RegistrationStore class will affect the search performance. In your README.txt, provide a few lines of explanation justifying your choice of data structure(s).
- The Driver code (use Driver class) should accept the following from the command line arguments: *inputDataFileName NN searchFileName MM DEBUG_VALUE*

- The value of MM should be between 1 and 5.
- The value of NN should be between 1 and 5.
- The value of DEBUG_VALUE should be between 0 and 4.
- The DEBUG_VALUE should be accordingly set in the class Debug. Use the DEBUG_VALUE in the following way:

- DEBUG_VALUE=4 [Print to stdout, using the Logger, everytime a constructor is called]
- DEBUG_VALUE=3 [Print to stdout, using the Logger, everytime a thread's run() method is called]
- DEBUG_VALUE=2 [Print to stdout, using the Logger, everytime an entry is added to the Results data structure]
- DEBUG_VALUE=1 [Print to stdout, using the Logger, the search results]
- DEBUG_VALUE=0 [No output should be printed from the application]

- Create a Logger.java class in the *util* package with a method

```
// int value is the DEBUG_LEVEL
// String is the user message
dump(int, String);
```

- The driver code should create a PopulateWorker instance and pass the following: (1) the number of threads to be spawned; and (2) a handle to another object that reads from a file. The PopulateWorker class should also *join* on the threads that are created.
- The driver code should create a SearchWorker instance and pass the following: (1) the number of threads to be spawned; and (2) a handle to another object that reads from a file. The SearchWorker class should also *join* on the threads that are created.
- The driver code should then print out the Results of the search.

Code Organization

- Your directory structure should be exactly as given in the code template

Code Templates

- Use the following code template.
- You can untar using the following commands on bingsuns (or any *nix machine)
  - gunzip csRegs.tar.gz
  - tar -xvf csRegs.tar
- You can comple and run the code with the following commands in the directory threadsStarter. Note that the file build.xml controls the processing of compiling, creating jar files, and running the code.
  - ant compile
  - ant run

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, you need to "tar" and then "gzip" your top level directory. Create a tarball of the directory lastName_firstName_assign1. We should be able to compile and execute your code using the makefile and instructions you provide in the README.txt file.
- Instructions to create a tarball
  - Make sure you are one level above the directory firstName_LastName_assign1.

- tar -cvf firstName_LastName_assign1.tar firstName_LastName_assign1/
- gzip firstName_LastName_assign1.tar

- For team projects, create the directory as lastName_firstName_lastName_firstName_assign1, so that both team members' names are used. Only one team member should submit.
- Do NOT send your assignment as an email attachment to be considered as submission.
- Upload your assignment to Blackboard, assignment-1.

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.
- Note: 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.
- Note: 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.).
- Note: 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

- The policy for late submissions is that you will lose 10% of the grade for each day that your submission is delayed.

Grading Guidelines

Grading Guidelines.

mgovinda at cs dot binghamton dot edu
Back to CS x42: Programming Design Patterns