Assignment 2

Due date

- 11.59 PM EST, June 28th.

Git

- Git url: https://classroom.github.com/a/VFqaNW0r

Submit your code as per the provided instructions. A signup sheet will be provided, if needed, to setup an appointment with the TA to provide a demo of your project via Skype or Hangout.

Updates

Posted a template for MyLogger.java

Assignment Goal

Application of the State Pattern.

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 anyone else. 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

- Use the State pattern to implement traffic lights at a 4-way intersection. You should have a minimum of 4 states. You can make simplifications so that the assignment can be completed on time! For example:
    - there isn't any protected-left
    - no-yellow lights
    - the input file has at least 4 vehicles that have reached the intersection on each side.
    - only 2 cars can go through the green light before it changes to red.
    - The input file can have lines to say which of the lights are green and red (north, south, east, or west).
    - The input file can have lines to say that a new vehicle has arrived in north side (similarly for the others),
    - The input file format should be easy to understand and also easy to parse.
- Use an input file that is read by line by line to determine the next event that the State diagram needs to deal with. Design your input format, and explain it in the README.md file.
- Design a single threaded State machine (don't use multiple threads).
- List your assumptions in your README file.

Input File and Output File

- Design an input file that is read line by line to get the actions to be called on the Context class. You can design the input file to be as simplified as you want. However, note that grading points have been allocated for submitting an input file that comprehensively checks your State machine.
- Unlike in other assignments, you are required to submit your input file as part of your submission.
- Create an ouput.txt and in it print meaningful statements from each method in the State classes so that the output.txt can be reviewed to determine if the State Machine worked correctly for the provided input file. We will test your code with your own input file.
- Add more enum values in MyLogger class for the various sections of your code. Instead of System.out.println(...), use MyLogger.writeMessage(....) throughout your code. Here are examples of how to use the Logger.
- `MyLogger.write(1, "entering readLine method in FileProcessor");`
- `MyLogger.write(7, "changing current state in the Context class");`

  Now, depending on the debug value passsed from the command line, the debug statemetns will only be printed from the FileProcessor, Context class, constructors, etc. You can design your scheme to decide which debug value should be used for each section/class of the code.

- Upload a file showing your State diagram in a well known format (pdf, jpeg, etc.)

Command Line Validation

- Accept a debug value as the last argument from command line. Set this debug value that is read from the command line to the staic value in MyLogger using the setDebugValue(...) method in MyLogger.java
- In your README.md, explain the cases for which command line validation has been provided (number of arguments, input.txt not found, etc.).

Code Organization

- Use code organization that is similar to Assignment-1.

```
john_doe_assign_2/
john_doe_assign_2/README.md
john_doe_assign_2/fourWayStreetLights
john_doe_assign_2/fourWayStreetLights/src
john_doe_assign_2/fourWayStreetLights/src/build.xml
john_doe_assign_2/fourWayStreetLights/src/BUILD
john_doe_assign_2/fourWayStreetLights/src/BUILD/classes
john_doe_assign_2/fourWayStreetLights/src/fourWayStreetLights
john_doe_assign_2/fourWayStreetLights/src/fourWayStreetLights/driver
john_doe_assign_2/fourWayStreetLights/src/fourWayStreetLights/driver/Driver.j
ava
john_doe_assign_2/fourWayStreetLights/src/fourWayStreetLights/util
john_doe_assign_2/fourWayStreetLights/src/fourWayStreetLights/util/FileProces
sor.java
john_doe_assign_2/fourWayStreetLights/src/fourWayStreetLights/util/Logger.jav
a               // Attempt to use it if you understand its design
john_doe_assign_2/fourWayStreetLights/src/fourWayStreetLights/util/FileDispla
yInterface.java
john_doe_assign_2/fourWayStreetLights/src/fourWayStreetLights/util/Results.ja
va              // Can again use Results to store all the output
john_doe_assign_2/fourWayStreetLights/src/fourWayStreetLights/util/StdoutDisp
layInterface.java
john_doe_assign_2/fourWayStreetLights/src/fourWayStreetLights/service/
john_doe_assign_2/fourWayStreetLights/src/fourWayStreetLights/service/StretLi
ghtsContext.java // Context class
john_doe_assign_2/fourWayStreetLights/src/fourWayStreetLights/service/StreetL
ightsStateI.java  // State Interface
john_doe_assign_2/fourWayStreetLights/src/fourWayStreetLights/service/StartSt
ateImpl.java  // Impl class for one of the states. Add more such classes
```

Submission

- Make sure to first run "ant clean" to remove the BUILD folder
- Delete the output file from your folder
- At the level above john_doe_assign_2 run the following commands
- `tar -cvf john_doe_assign_2.tar john_doe_assign_2/`
- `gzip john_doe_assign_2.tar`
- Alternatively, at the level above john_doe_assign_2 run the following commands
- `tar -czvf john_doe_assign_2.tar.gz john_doe_assign_2`

- 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.md /UL>

  ### General Requirements

  - Start early and avoid panic during the last couple of days.
  - Submit a README.md file (placed at the top level).
  - 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