CST316 Software Enterprise II: Construction and Transition **Spring 2016 ONLINE**
Lab Code Inspection Exercise **DUE February 8, 11:59pm**

Objectives: Understand how review code and to integrate reviews into source code control.

Task 1: Perform Code Review (30%)

1. For this lab grab the code zipfile off Blackboard. Expand the zipfile in an empty folder on your computer.
2. Create a GitHub repo with your copy of the code. Name the repo cst316_codereviewlab_<asurite>
3. Review the source code files individually for conformance to the coding standards provided on the next page, for general quality practices, and for logic errors. Log any defects using the MS-Word form. Make sure your individual name is at the top and the form is named codereviewlab_<asurite>.docx, and save it.
    - On the Word form at the top is a place to enter a GitHub repo. Put your GitHub repo name there. We need this so we know where to go to find your GitHub work to grade the lab.

Task 2: Apply the Git Pull Workflow (70%)

- You should have reviewed the Git Pull Workflow content under Sprint 2 on the Project page on Blackboard. This content describes a workflow where you create a branch per each user story. Well, in this task we will create a branch for each defect.

1. Select any 2 non-stylistic defects from your Word document in Task 1. Keep in mind the defect number on the form.
2. Create 2 remote branches, 1 for each defect. Name them bug-# where #s are the number identifiers of the defects on your code review form.
3. Pick one bug and resolve the defect locally on your computer, in the correct branch.
4. Push the change to the proper remote branch on GitHub.
5. Create a pull request for the change back from the bug branch back to master.
6. Accept the pull request, entering a comment “Task 2 pull approved”
7. Repeat steps 3-5 for the second bug. However this time do not accept the pull request (step 6), instead close it with a comment “Task 2 pull request denied”.

DO NOT DELETE YOUR GITHUB BRANCHES AFTER CLOSING THE PULL REQUESTS!!! WE NEED THESE TO GRADE YOUR LAB!!!

Grading: The code given compiles and runs but has several faults. To motivate you to find and log defects, your grade will be in part determined by how many of the defects you find that I have seeded throughout the code. Another part is the extent to which you improve the internal code quality of the code. Below is a Coding Standard the code should follow, but keep in mind violation of the coding standards are only one type of defect (see the Category description at the bottom of the Code Review Defect List).

Expectation for your projects going forward:

I expect that from this point forward you will incorporate code reviews into your quality policy:

1. Decide on when and where and why you will do reviews. Which code? How many people? What level of formality? *I expect each team to define one formal and one informal code review process in their quality policy and post this on their Taiga wiki.*
2. Define how you will categorize issues – level of severity, type of defect, etc. How will you indicate that?
3. Define a coding standard. Quality codebases look like one person writes them.
4. Integrate your process into GitHub. I expect to be able to track code reviews you have performed via GitHub. I should clearly see pull requests with labels/comments based on code reviews. Follow the Git Pull Workflow.
5. Make sure in your process it is clear whose code is getting reviewed, and by whom it is getting reviewed. You should serve as both an author and a reviewer to get credit for this module on your project.

Coding Standards

1. All source code files must have a file banner comment present and filled in. This banner is available in the templates.java file on Blackboard.
2. All public classes must have a class banner comment present and filled in. This banner is available in the templates.java file.
3. All public methods, except getter/setter methods, must have a method banner comment present and filled in. This banner is available in the templates.java file.
4. Naming conventions are as follows:
    a. Constants and Enums should be in all CAPS (example: PI)
    b. Class names should be upper CamelCase, with the first letter uppercase (example: MyClass).
    c. Variable, Parameter, and Method names should be in lower camelCase, with the first letter in lowercase (example: fooBar).
    d. Non-public methods, variables, and constants should be prefixed by a leading underscore _.
5. All attributes must be private (class member variables, not constants).
6. All literal values, except loop indices starting at 0 or 1 must be declared as constants.
7. Classes should list methods and attributes in the following order, top-to-bottom:
    a. Constants
    b. Constructors
    c. Public setter/getter methods in alphabetic order
    d. Public non-getter/setter methods
    e. Private methods
    f. Private class member variables
8. All code should be consistent stylistically. This includes:
    a. All {} should appear with the { at the end of a line and } on its own line
    b. Indentation should be consistent.
    c. All complex statements (if, else, switch, loops) must use explicit {} even if the body is a single line.