CS 520
Homework 2
Implementation & testing

Due: **Thursday April 1, 2021, 9:00 AM** via Moodle. You may work with others on this assignment but each student must submit their own write up, clearly specifying the collaborators. The write ups should be individual, not created jointly, and written in the student's own words. Late assignments will be accepted for extenuating situations.

Overview and goal

The goal of this assignment is to redesign, reimplement, and test a Row game, according to the model-view-controller (MVC) architecture pattern.

The following repository provides a basic implementation of the Row game:
https://github.com/LASER-UMASS/cs520-Spring2020

This quick-and-dirty implementation satisfies some best practices but violates other best practices. It needs a major design overhaul. In contrast to the current version, your implementation should support possible extensions aiming to satisfy the open/closed principle (or at least improve encapsulation). Additionally, your implementation should enable individual components to be tested in isolation.

You are expected to clone the existing repository and keep your implementation under version control, using the cloned repository. You will submit your repository to us, so you should make coherent and atomic commits (in particular at least for the 3 sections below), and use descriptive log messages.

How to get started

1. Clone the repository https://github.com/LASER-UMASS/cs520-Spring2020 containing the *threeinarow* folder
2. Read the provided *README* in the *threeinarow* folder.
3. Use the commands to document, compile, test, and run the application from that folder.
4. Familiarize yourself with the original application source code contained in the *src* folder: src/RowGameApp.java, src/controller/*.java, src/model/*.java, src/view/*.java.

Implementation (Approximately 2/3 of the total points)

Your version of the application must adhere to:

- the MVC architectural pattern (This has already been started but needs to be finished.)
- the Observer design pattern and also Strategy or Template method design pattern
- OO design and best practices

Best practice violations

Your reimplementation should reflect 3 proposed fixes of the violations of best practices (either your own or from the possible answers). If the proposed fix has already been implemented, simply add an internal comment in the appropriate source code to document that.

Observer design pattern

You're responsible for applying the *Observer design pattern*. From the RowGameApp class perspective, the *Observable* should be the *RowGameModel*, the *Observers* should be the *Views*, and the *update* method should be the *Views' update* methods.

For Alternative 1, apply the standard Observer design pattern

- Observer provides update method
- Observable provides getState, setState, and stateChange methods as well as register(Observer) and unregister(Observer)

For Alternative 2, apply one of the Swing versions of the Observer design pattern:

- https://docs.oracle.com/en/java/javase/12/docs/api/java.desktop/java/beans/PropertyChangeSupport.html
- https://docs.oracle.com/en/java/javase/12/docs/api/java.desktop/java/beans/PropertyChangeListener.html

To implement this pattern, the *RowGameModel* should use one of the above and the *Views* should use the other one.

For either Alternative, the RowGameController class should now only have a *RowGameModel* field but not have any *Views* fields. The *Observer design pattern* will then ensure that the *Observable* (*GameRowModel*) keeps the *Observers* (*Views*) up-to-date.

Strategy or template method design pattern

You're also responsible for applying the *Strategy or template method design pattern*.

Here are the basic rules of the *Three in a Row* game:

- Initially, the game board has each game block empty. The legal moves are in the bottom row.
- There are two players. Player 1 marks their blocks with 'X' while player 2 marks their blocks with '0'. Player 1 gets to make the first move.
- A legal move is to either an empty block in the bottom row or an empty block in an upper row on top of a filled block in the row immediately below.
- A player wins if they connect 3 of their marks (either 'X' or 'O') in a horizontal, vertical, or diagonal line. If neither player wins and all blocks are filled in, the game ends in a draw (or tie).
- After either player resets the game, the game goes back to its initial configuration.

Here are the basic rules of the *Tic Tac Toe* game:

- Initially, the game board has each game block empty. The legal moves are to any of the blocks.
- There are two players. Player 1 marks their blocks with 'X' while player 2 marks their blocks with '0'. Player 1 gets to make the first move.

- A legal move is to an empty block.
- A player wins if they connect 3 of their marks (either ’X’ or ’O’) in a horizontal, vertical, or diagonal line. If neither player wins and all blocks are filled in, the game ends in a draw (or tie).
- After either player resets the game, the game goes back to its initial configuration.

For Alternative 1, the RowGameController concrete class should have a RowGameRulesStrategy interface to use to follow the rules of either *Three in a Row* or *Tic Tac Toe*.

For Alternative 2, the RowGameController abstract class should have rule-related template methods. It should have two subclasses: one for *Three in a Row* and another for *Tic Tac Toe*.

For either Alternative, you could either have the RowGameApp take an input parameter specifying which rules to use. Alternatively, you could modify the view.RowGameUI to add a Menu and Menu Item to specify which rules to use.

Testing (Approximately 1/3 of the total points)

Your design must allow testing of individual components. To show testability, you are expected to submit at least fourteen (14) test cases in total. You may need to add more details to the 2 existing test cases as well as you will need to implement 12 new test cases.

You need at least one (1) test case per package (*model*, *view*, and *controller*) along with your implementation.

For each of the row game rules (*Three in a Row* and *Tic Tac Toe*), you also need the the following test cases:

- Illegal move (should not change the game board)
- Legal move (should change the game board)
- One of the players win
- The two players tie
- Reset

Deliverables (A few points)

Your submission, via Moodle, must be a single archive (.zip, .tar, or .tar.gz) file, containing:

1. An answers.txt or answers.pdf file (with your name and any collaborators at the top) describing the following:
   (a) For each violation of best practices, is the proposed fix implemented by your code or by the MVC refactoring
   (b) For the RowGameController, which design pattern did you apply? Strategy or Template method
   (c) In the model package, you applied the Observer design pattern. Which class is the Observable: RowGameModel, RowBlockModel, or both?
2. The *cs520-Spring2020* folder with all the updated source files and test cases of your application residing inside the *threeinarow* folder. Make sure the *.git* folder exists in the *cs520-Spring2020* folder in which you committed your code. You can see your commits by running the *git log* command inside the *cs520-Spring2020* folder. The repository should have a set of coherent commits showing your work, not a single version of the code.

3. A *README* file describing the **commands including their arguments to compile, test, and run** your code from within the *threeinarow* folder. You can use *Ant* or other build tools, but the *README* needs to explicitly say for each of the three commands how to specify its command line arguments (refer to the existing *README* provided in the *threeinarow* folder).

Your application is expected to compile and run correctly for *Three in a Row* and *Tic Tac Toe*. Unless your application can be compiled and tested wth the provided build file, please provide brief instructions for how to compile, test, and run your code with a *README* file that should exist inside the *threeinarow* holder.