CS 520
Homework 3
Design patterns, testing, & debugging

Due: **Tuesday November 30, 2021, 11:59 PM** 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 (both natural language and code) 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 following repository provides a basic implementation of the Row game app:
https://github.com/LASER-UMASS/cs520.git

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 2 sections below), and use descriptive log messages.

How to get started

1. Clone the repository https://github.com/LASER-UMASS/cs520.git containing the *tictactoe* folder with "git clone -b v2.0.0 https://github.com/LASER-UMASS/cs520.git"
2. Read the provided *README* in the *tictactoe* 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.

Your version of the application should adhere to:

- the MVC architectural pattern
- the Observer design pattern as well as either the Strategy design pattern or else the Template Method design pattern
- Best programming practices
- OO design principles and patterns

Design patterns [Approximately 2/3 of the points]

Observer design pattern [Approximately 1/3 of the points] 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 [Approximately 1/3 of the points] You're also responsible for applying the *Strategy or template method design pattern*.

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.

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.

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.

Internal documentation

- You should update the README file to document how to switch between the two game rules.
- You should generate the javadoc (contained in the jdoc folder) and commit it.

Debugging [Approximately 1/3 of the points]

You should select a Java Integrated Development Environment (e.g., Eclipse, VSC). In that environment, you should build the Row game app and then run the JUnit test runner for it. For the debugging, you should do the following:

1. Add a breakpoint for the resetGame method
2. In the Debugger, run the Row game app
3. Perform a legal move
4. Reset the game

Here are the screenshots that should be included:

- Javadoc view
- JUnit test runner showing all of your test cases passing
- Debugger showing the breakpoint for the resetGame method
- Debugger showing the program execution state after calling the move method but before calling the resetGame method (the legal move block contents should be filled in)
- Debugger showing the program execution state after calling the resetGame method (all of the block contents should be empty again)

Deliverables [Less than 10 points]

Your submission, via Moodle, must be a single archive (.zip or .tar.gz) file named hw3, containing:

1. The 5 screenshots from the Debugging section
2. The *cs520* folder with all the updated source files and test cases of your application residing inside the *tictactoe* folder. Make sure the *.git* folder exists in the *cs520* folder in which you committed your code. You can see your commits by running the *git log* command inside the *cs520* 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 *tictactoe* 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 *tictactoe* 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 *tictactoe* holder.