CS 520
Homework 2
Design, implementation, & testing

Due: **Saturday October 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 goal of this assignment is to redesign, reimplement, and test a Tic Tac Toe game, according to the model-view-controller (MVC) architecture pattern.

The following repository provides a basic implementation of the Tic Tac Toe game:
https://github.com/LASER-UMASS/cs520.git

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 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 v1.0.0 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 must adhere to:

- the MVC architectural pattern (This has already been started but needs to be finished.)
- Best programming practices
- OO design principles and patterns

Implementation (Approximately 3/4 of the total points)

Best practice violations (Approximately 1/2 of the total points)

Your reimplementation should reflect 4 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. You should also generate the javadoc (contained in the jdoc folder) and commit it.

Design patterns (Approximately 1/4 of the total points)

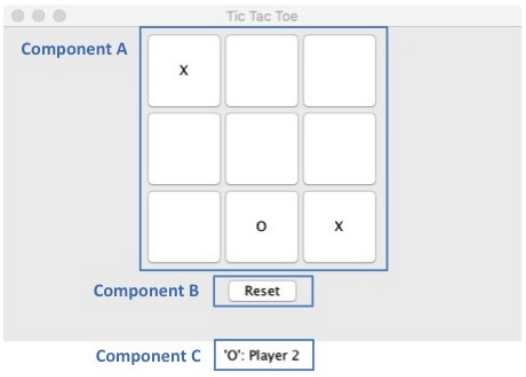

Figure 1: Main components of the 'TicTacToe' UI

Composite design pattern For the MVC architecture pattern, The RowGameGUI class represents the View. For the *Composite* design pattern, here is the proposed *Component* class.

```
package view;

import model.RowGameModel;

public interface RowGameView
{
public void update(RowGameModel gameModel);
}
```

You should first briefly describe how to decompose the RowGameGUI class into the following two classes:

1. RowGameBoardView (i.e. Component A)
2. RowGameStatusView (i.e. Component C)

For each class, which fields? What does the update method do?

You should then briefly describe how to modify the RowGameGUI class to be a *Composite* class that uses the above to classes.

Observer design pattern For the original application, the *Observer* design pattern is being applied for the relationship between a Java Swing View and its Controller (in particular for Component A and Component B).

- Identify one field in the original application that corresponds to an *Observable*.
- For that *Observable*, identify the Java Swing class that corresponds to its *Observer*.
- For that *Observer*, identify the implementation of the *update* method.

Testing (Approximately 1/4 of the total points)

Your design must allow testing of individual components. To show testability, you are expected to submit at least ten (10) 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 8 new test cases.

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

For the *Tic Tac Toe*) game rules, 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 (Less than 10 points)

Your submission, via Moodle, must be a single archive (.zip or .tar.gz) file named hw2, containing:

1. An answers.txt or answers.pdf file (with your name and any collaborators at the top) describing the following:
   (a) For each of the 4 violations of best practices, is the proposed fix implemented by your code or by the MVC refactoring
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 *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.