CS 520
Homework 4
OO design & debugging

You may work with others on this assignment. Each programming pair, however, must submit their own codebase, clearly specifying the collaborators. The codebase should be written in the pair's own words.

A programming pair may contain 1, 2, or 3 students. This allows the usual groups of 3 to remain together.

You should submit on Gradescope. Late assignments will be accepted with **prior** permission or for extenuating circumstances. Due: May 10, 2023, 11:59 PM

There are a total of 100 points.

Overview

The goal of this assignment is to design, implement, and test a Tic Tac Toe app, to improve adherence to non-functional requirements (e.g., understandability, modularity, extensibility, testability), design principles/patterns, and best practices.

This implementation applies the MVC architecture pattern. In contrast to the current version, your implementation should support possible extensions aiming to satisfy the open/closed principle (specifically information hiding with 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 for the first 3 sections below), and use descriptive log messages.

How to get started

1. Clone the third version of the repository https://github.com/LASER-UMASS/CS520.git containing the *tictactoe* folder with "git clone https://github.com/LASER-UMASS/CS520.git -b v1.2.0" (This is our hw3 solution.)

   NOTE) Alternatively, you may build on your hw3 solution.

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* and *test* folders:

   - src/RowGameApp.java
   - src/controller/RowGameController.java
   - src/view/GameBoardView.java.java
   - src/view/GameStatusView.java
   - src/view/RowGameGUI.java

    - src/view/UndoViewController.java
    - src/view/View.java
    - src/model/BlockIndex.java
    - src/model/RowGameBlock.java
    - src/model/RowGameModel.java
    - test/TestExample.java

Your version of the application must adhere to:

- the MVC architecture pattern
- Design principles and patterns
- Best programming practices

Understandability: Documentation [Approximately 1/10 of the points]

- You should update the README file to document any new functionality.
- You should generate the javadoc (contained in the jdoc folder) and commit it.
- You should have incremental commits about your modifications.

Extensibility: Human or computer player [Approximately 3/10 of the points]

The user should be able to select between playing against another human player or a computer player.

For this extension, the RowGameController class needs to apply either the Strategy design pattern or template method design pattern. Specifically the helper method is the move method.

If the user is playing against a human player, you should be able to use your original move method.

If the user is playing against a computer player, you will need to implement another move method:

- The user should be Player 1 and the computer should be Player 2.
- After the user performs one move and the game is not over, the computer player should then perform their move.

The extension will need to be tested (see the Testability section).

Testability: Unit test suite(s) [Approximately 1/10 of the points]

You should regression test to make sure that the existing 9 test cases still pass. Additionally, you should add one (1) new test case:

1. Against a computer player: After performing a legal move, the game is updated appropriately.

Each new test case needs to apply the test case template from the test driven development lecture.

Debuggability: Debugger [Approximately 2/5 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. This should be in the graphical UI and not the command line UI.

For the debugging, you should do the following:

1. Add a breakpoint for the undo
2. In the Debugger, run the Row game app
3. Move
4. Undo

Here are the five (5) screenshots that should be included:

1. Java API view (e.g., javadoc, class outline)
2. JUnit test runner showing all of your test cases passing
3. Debugger showing the breakpoint for the undo
4. Debugger showing the program execution state (usually the variables view) after calling the move but before calling undo (show the model and/or text components set appropriately)
5. Debugger showing the program execution state (usually the variables view) after calling the undo (show the model and/or text components are empty again)

Deliverables [Approximately 1/10 of the points]

Your submission, via Gradescope, must be a single archive (.zip) file named hw4, containing:

1. The *tictactoe* folder with all the updated source files and test cases of your application. The git log needs to be submitted as either a text or PDF file. The git log should have a set of coherent commits showing your work, not a single version of the code. Because Gradescope has issues with very large folders containing many files, you should NOT include the .git folder in your submission.
2. A PDF file named debugging.pdf containing the 5 screenshots from the debugging section

We will be checking the following:

1. The git log (either text or PDF file) is included and has incremental commit messages.
2. The app compiles and runs.
3. The test suite compiles, runs, and all test cases pass.