CPSC 224 Software Development

Gonzaga University

Gina Sprint

PA5: Layouts and Dialogs (100 points)

Individual, non-collaborative assignment

Learner Objectives

At the conclusion of this programming assignment, participants should be able to:

- Organize components using a variety of layout managers
    - Nested layouts
    - `GridLayouts`
- Visually group a layout's components by adding a border to panel
- Create dialogs

Prerequisites

Before starting this programming assignment, participants should be able to:

- Build a Swing GUI using a `JFrame` and `JPanel`
- Utilize common Swing components such as `JTextField`, `JLabel`, and `JButton`
- Organize components in `BoxLayouts`
- Handle button click events using the `ActionListener` interface

Acknowledgments

Content used in this assignment is based upon information in the following sources:

- Nadra Guizani's 224 assignment

Github Classroom Setup

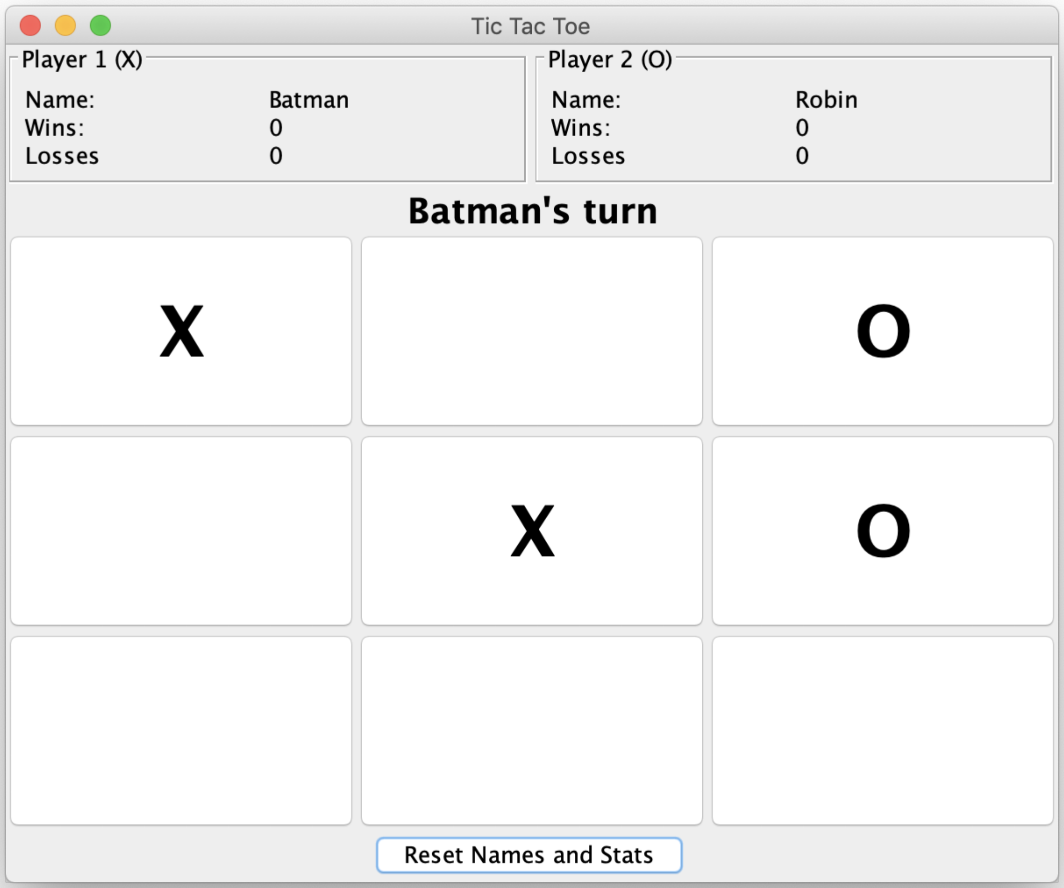
For this assignment, you will use GitHub Classroom to create a private code repository to track code changes and submit your assignment. Open this PA5 link to accept the assignment and create a private repository for your assignment in Github classroom: https://classroom.github.com/a/9ZP8AUBW

Your repo, for example, will be named GonzagaCPSC224/pa5-yourusername (where yourusername is your GitHub username). I highly recommend committing/pushing regularly so your work is always backed up. We will grade your most recent commit, even if that commit is after the due date (your work will be marked late if this is the case).

Overview and Requirements

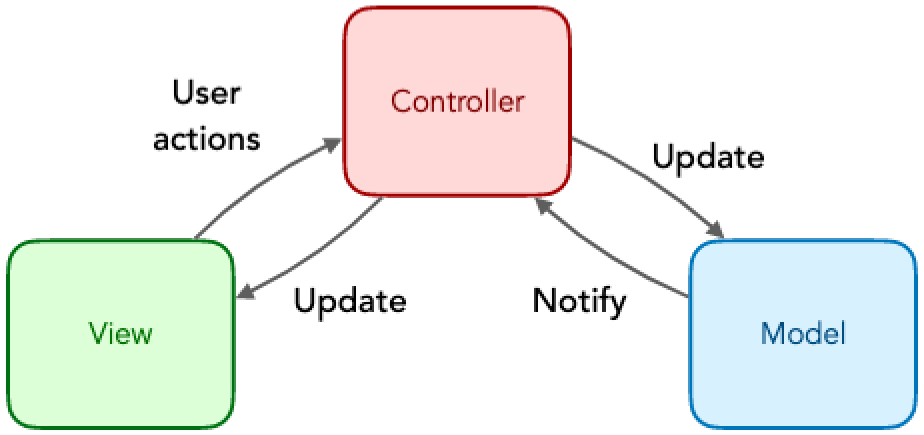
We are going to make an app for two users to play Tic Tic Toe. In PA2, we derived the logic for a console-based Tic Tac Toe program. With the MVC (model-view-controller) architecture, our PA2 `TicTacToeBoard`, `Coordinate`, and `Cell` classes together represent the model of our Tic Tac Toe app. We can copy and paste these files into our /src folder in IntelliJ IDEA. We reuse these classes in a Swing GUI with minimal modifications (including defining a fourth class, `TicTacToeModel`).

With the model of our app (mostly) already coded, our task for PA5 is to design the associated Tic Tac Toe view (a `JFrame`) and controller so our users can play our game with a GUI (graphical user interface). For example, your GUI will look something like this:



Classes to Define

It is good practice to separate the view of the app (`TicTacToeView.java`), from the controller of the app (`TicTacToeController.java`), from the model of the app (`TicTacToeModel.java`). Such a design is called Model-View-Controller (MVC) architecture. Using MVC makes our app easier to design, modify, and maintain now and later on.



(image from https://www.cocoawithlove.com/blog/mvc-and-cocoa.html)

Define three classes to implement a rudimentary MVC architecture:

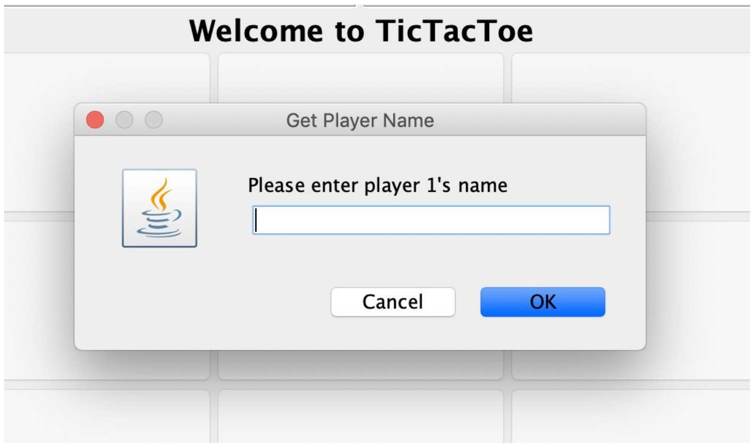
1. `TicTacToeModel`: contains all of the functionality for the game of Tic Tac Toe
   a. You've already solved this for PA2 as a procedural solution with three classes. Now, you need to update this solution to have a clean interface for the controller to use. You'll also need to design the model with encapsulation in mind (think access modifiers!!)
   b. `TicTacToeModel` does not directly communicate with `TicTacToeView`, it is view independent!!
   c. Expose functionality for `TicTacToeController` to access.
2. `TicTacToeView`: contains all of the view initialization
   a. `TicTacToeView` inherits from `JFrame`
   b. `TicTacToeView` does not directly communicate with `TicTacToeModel`
3. `TicTacToeController`: contains the "glue" to communicate between the `TicTacToeModel` and the `TicTacToeView`
   a. Contains your `main()` method that instantiates a `TicTacToeModel` and a `TicTacToeController`
   b. Responds to changes in the GUI
      i. E.g. it implements `ActionListener`
   c. Receives updates from the `TicTacToeModel` and updates the GUI

Note: I don't expect your MVC architecture to be perfect your first time (or even the second time, or even so on….). MVC is tough to understand, which is why we are starting early! I want you to try and separate your model from your view by defining the above three classes. The goal in doing so is you could swap in a different view if you wanted (e.g. an Android app GUI or a JavaFX GUI) and you wouldn't have to change your model because you defined minimal dependencies in your code.

Additional Requirements

Your Tic Tac Toe GUI app should match the above screenshot as close as you can

- You can set the size of the frame as you see fit
- Use layout managers as you see fit. At a minimum though...
    - You must have a `GridLayout` of 3x3 `JButtons` to represent the board
        - Set the `JButtons` to be disabled when appropriate (think about what these situation(s) would be…)
        - The font for each `JButton` text should be 40-pt and bold
    - You must have two `GridLayouts`, one for each player and their info (names, wins, and losses)
        - Surround each player info with a title border
        - See screenshot above for an example
- Inform the user of game updates via a "status label"
    - Mine is above the grid of buttons
    - Use it to show whose turn it is, invalid moves, and relevant game state
- Prompt the user for the player names via **input dialogs** at the beginning of the game and when they press the "Reset Names and Stats" button

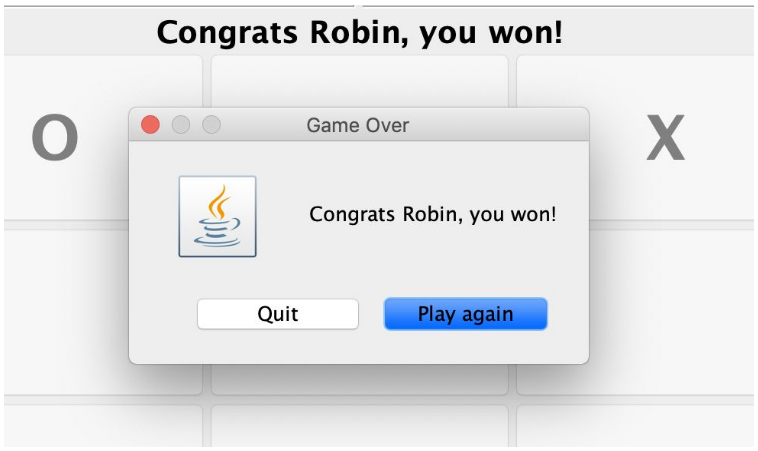


Welcome to TicTacToe

Get Player Name

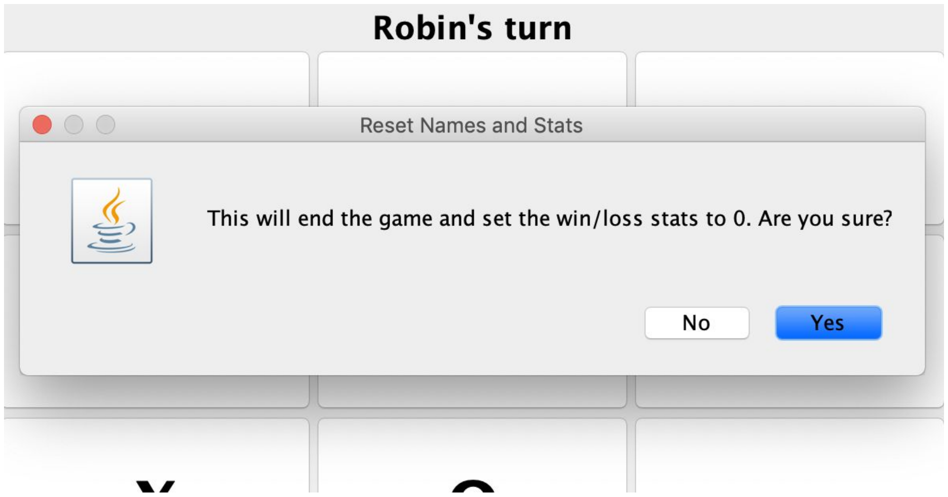
Please enter player 1's name

    - Note: be sure to provide default names ("Player 1" and "Player 2") if the user doesn't provide names
- Prompt the user if they want to play again or quit at the end of each game via an **option dialog**


Congrats Robin, you won!

Congrats Robin, you won!

- Allow the user to reset the player names, player stats, and start a new game at any time by pressing the “Reset Names and Stats” button
    - Prompt the user via a **confirmation dialog** to confirm they really want to reset


Robin's turn

This will end the game and set the win/loss stats to 0. Are you sure?

Include a **UML sequence diagram** of your solution in your top-level project folder. You can hand draw this and then take a picture/scan it or you can create it digitally using a tool like https://www.draw.io/

Bonus (5 pts)

Keep track of wins/losses for players between executions of the app:

- Prompt the user when they close the app if they’d like to save the current player stats.
- Prompt the user when they open the app if they’d like to load player stats.

You’ll need to devise a way to store/retrieve this information to/from files. Handle special cases appropriately!!

Submitting Assignments

1. Use Github classroom to submit your assignment via a Github repo. See the “Github Classroom Setup” section at the beginning of this document for details on how to do this. You must commit your solution by the due date and time.
2. Your repo should contain your entire IntelliJ IDEA project. Double check that this is the case by cloning (or downloading a zip) your submission repo and opening your project in IntelliJ IDEA and running your code.

Grading Guidelines

This assignment is worth 100 points + 5 points bonus. Your assignment will be evaluated based on a successful compilation and adherence to the program requirements. We will grade according to the following criteria:

- 15 pts for implementing the requirements of the `TicTacToeModel` class, including appropriate encapsulation
- 40 pts for implementing the requirements of the `TicTacToeController` class
  - 5 pts for updating the status label
  - 5 pts for showing a dialog when the game ends and handling the user’s dialog choice
  - 10 pts for correct grid button click event handling and updating buttons
  - 10 pts for correct reset button click event handling and confirmation dialog use
  - 5 pts for showing input dialogs to get player names when the app first launches and on reset
  - 5 pts for handling the cases where the user doesn’t enter a name in the input dialog
- 25 pts for implementing the requirements of the `TicTacToeView` class
  - 10 pts for player info `GridLayouts` with borders
  - 10 pts for grid of `JButtons` that fills the available screen space
  - 5 pts for matching the remaining layouts to the screenshot provided (status label, reset, etc.)
- 10 pts for including an accurate UML sequence diagram
- 10 pts for adherence to proper programming style and comments established for the class