CPSC 224 Software Development

Gonzaga University

Gina Sprint

PA4: GUI Basics (100 points)

Individual, non-collaborative assignment

Learner Objectives

At the conclusion of 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

Prerequisites

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

- Understand and implement object-oriented programming concepts in Java

Acknowledgments

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

- None to report

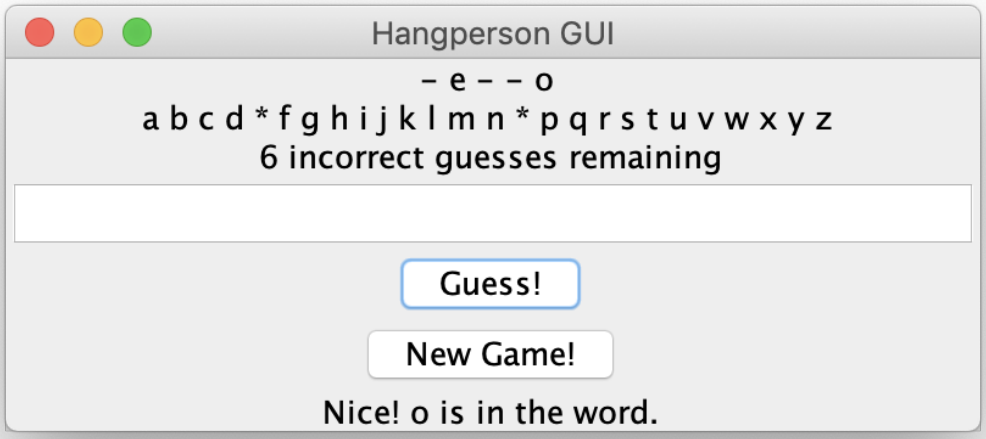
Github Classroom Setup

For this assignment, you will use GitHub Classroom to create a private code repository to track code changes and submit your assignment. Open this PA4 link to accept the assignment and create a private repository for your assignment in Github classroom: https://classroom.github.com/a/1-9ctPy3

Your repo, for example, will be named GonzagaCPSC224/pa4-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).

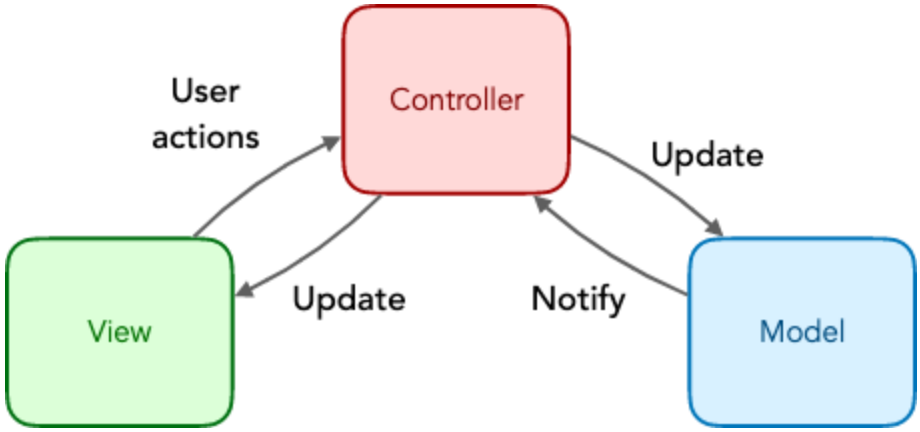
Overview and Requirements

We are going to make a graphical user interface (GUI) for our hangman game from PA1-Java-Basics!! For example, your GUI will look something like this:



Classes to Define

It is good practice to separate the view of the app (`HangpersonView.java`), from the controller of the app (`HangpersonController.java`), from the model of the app (`HangpersonModel.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. `HangpersonModel`: contains all of the functionality for the game of hangperson

    a. You've already solved this for PA1 as a procedural solution with static methods. Now, you need to re-write this solution to be a clean, OOP solution with encapsulation (think access modifiers!!)
    b. `HangpersonModel` does not directly communicate with HangpersonView, it is view independent!!
    c. Expose functionality for HangpersonController to access.
2. `HangpersonView`: contains all of the view initialization
    a. `HangpersonView` inherits from `JFrame`
    b. `HangpersonView` does not directly communicate with `HangpersonModel`
3. `HangpersonController`: contains the "glue" to communicate between the `HangpersonModel` and the `HangpersonView`
    a. Contains your `main()` method that instantiates a `HangpersonModel` and a `HangpersonView`
    b. Responds to changes in the GUI
        i. E.g. it implements `ActionListener`
    c. Receives updates from the `HangpersonModel` 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 Hangperson GUI app should match the above screenshot as close as you can

- Use a vertical `BoxLayout`
- You can set the size of the frame as you see fit
- Show a string representation of `visibleLetters`, `availableLetters`, and the number of remaining guesses
- Use a `JTextField` to get user input, which you still need to validate!!
    - Make sure a single lower-case letter is entered
    - Your app should not crash for any cases when the user presses the "Guess!" button 🤓
    - Inform the user if they enter invalid input via the status label...
- Inform the user game updates via a "status label"
    - Mine is at the bottom of the app
- Allow the user to start a new game at any time by pressing the "New Game!" button
    - If there are no more words to guess then inform the user
- Set the JButtons to be disabled when appropriate (think about what these situation(s) would be…)

- Hint for one case: `guessButton.setEnabled(false);` when they shouldn't be allowed to guess anymore

Your array of words to choose from should be read in from a file (instead of hardcoding them).

- There should be at least 10 words in the file, one on each line.
- The input file should be called words.txt and should exist on the user's desktop.
  - Do not hardcode the path to words.txt on your desktop! Your code should work for any currently logged in user.
  - Here is code to help you get started with this:

```
Path path = Paths.get(System.getProperty("user.home"),"Desktop",
"words.txt");
```

Include a UML class 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)

Use an `ImageIcon` and a `JLabel` to show an image representing the user's hangman progress. For example:

You can use the hangman images available from here: https://www.oligalma.com/en/downloads/images/hangman or you can create your own! Be sure to cite your source if you decide to use images that are not your original work. Put your images in a folder called images in your top-level project folder.

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:

- 25 pts for implementing the requirements of the `HangpersonModel` class
    - 5 pts for reading words from words.txt
    - 10 pts for converting your static procedural solution to an OOP solution
    - 10 pts for incorporating encapsulation and exposing only what needs to be accessed by `HangpersonController`
- 25 pts for implementing the requirements of the `HangpersonController` class
    - 10 pts for updating GUI components
        - Including updating a status label informing the user for all game state changes
    - 5 pts for correct "Guess!" button click event handling
    - 5 pts for correct "New Game!" button click event handling
    - 5 pts for handling special case(s)...
        - So your app does not crash
- 25 pts for implementing the requirements of the `HangpersonView` class
    - Match the layout to the screenshot provided
- 15 pts for including an accurate UML class diagram
- 10 pts for adherence to proper programming style and comments established for the class