**Assignment 5: Software Design**

In this assignment you will design a simple word game that may be used by a single player. The detailed requirements for the application are provided below.

To create your design, you should follow the same approach that we present in the P3L2 lesson. That is, analyze the requirements to identify and refine **(1) classes, (2) attributes, (3) operations, and (4) relationships** in your design. Just to be completely clear, **your task is to design the system, not to implement it**. The requirements for the application are listed below, in the “Requirements” section. Please note that not every requirement will be fully and directly represented in your design. For instance, at this level of detail, you do not have to show any purely GUI specific classes, if they are only doing user display and input and not performing any significant business logic. Similarly, any database support layer may be left out, if it is purely doing persistence tasks on **data and collections fully represented in the diagrammed layer**.

Your design should be expressed using a UML class diagram, and the level of detail of the design should be analogous to the level of detail we used throughout the **whole** P3L2 lesson (i.e., do not limit your design to only the elements focused on in the final video, but consider the entire process). Specifically, **you must provide enough details for the design to be self contained and for us to be able to assess whether the design suitably realizes all system requirements**.

To help with this, you must also provide a “design information” document, in which you concisely describe, **for each of the requirements listed below**, how that requirement is either realized in your design, or why it does not directly affect the design and is not shown. **Copy the list of requirements, and add your explanation for each one of them**. For example, using some partial requirements for a cash register application:

…

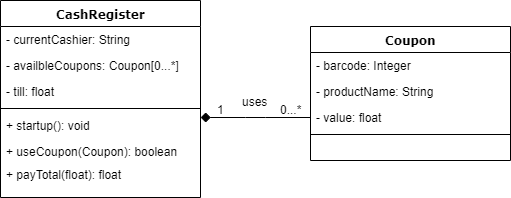
2. After starting the cash register, the cashier enters her name, and the total amount of money available in the till.

To realize this requirement, I added a ‘currentCashier’ to the register class to track the signed in cashier, and a float ‘till’ to represent the money in the till. These values are entered by the startup() method, after prompts are handled within the GUI.

…<additional requirements reflected in example UML diagram>...

16. The User Interface (UI) must be intuitive and responsive.

This is not represented in my design, as it will be handled entirely within the GUI implementation.



This explanation should be clear enough to allow us to follow the rationale behind your design and **how it will fulfill each specified requirement, including any that are not directly depicted in your class diagram.** You can also provide in the document additional information about your design, such as assumptions or rationale for some design decisions. Use the document to review your design and ensure you have included everything necessary for your application to fill the list of requirements.

You can use any UML tool you prefer, but do not hand draw your design. If you are not familiar with any specific tool, we recommend that you ask on Piazza for suggestions. There are usually some student discussions and recommendations for that on Piazza.

# Requirements

1. When the application is started, the player may choose to (1) Play a word game, (2) View statistics, or (3) Adjust the game settings.
2. When choosing to *adjust the game settings,* the player (1) may choose for the game to end after a certain *number of minutes,* from 1 to 5, defaulting to 3, (2) may adjust the *size of the square board,* between 4(x4) and 8(x8)*,* defaulting to 4, and (3) may adjust the *weights* of the letters of the alphabet between 1 and 5, defaulting to 1.
3. When choosing to *play a word game,* a player will:
   1. Be shown a ‘board’ of randomly generated letters.
   2. Be shown a timer counting down the *number of minutes* available for the game, as set in the settings.
   3. Start with 0 points, which is not required to be displayed during the game.
   4. Until the game timer counts to zero and the game ends*:*
      1. Enter a unique word made up of two or more letters on the board. The word must contain only letters from the board that are each adjacent to the next (horizontally, vertically, or diagonally) and a single letter on the board may not be used twice. The word does not need to be checked against a dictionary (for simplicity, we will assume the player enters only real words that are spelled correctly).

or

* + 1. Choose to re-roll the board at a cost of 5 points. The board will be re-created in the same way it is generated at the start of each game, replacing each letter. The timer will not be reset or paused. The player’s score may go into negative values.

or

* + 1. Choose to exit the game early.
  1. At the end of the game, when the timer has run out or the player chooses to exit, the final score for the game will be displayed.
  2. Each word will score 1 point per letter (‘Qu’ will count as 2 letters), and the cost for each board reset will be subtracted.
  3. After the player views the score, they will continue back to the main menu.

1. Whenever the board is generated, or re-generated it will meet the following criteria:
   1. The board will consist of a square full of letters. The square should have the number of letters, both horizontally and vertically, indicated by the *size of the square board* from the game settings (4x4, 5x5, 6x6, 7x7, or 8x8).
   2. ⅕ (rounded up) of the letters will be vowels (a,e,i,o,u). ⅘ will be consonants.
   3. The letter Q will be displayed as ‘Qu’ (so that Q never appears alone).
   4. The location and particular letters should be randomly selected from a distribution of letters reflecting the *weights* of letters from the settings. A letter with a weight of 5 should be 5 times as likely to be chosen as a letter with a weight of 1 (assuming both are consonants or both are vowels). In this way, more common letters can be set to appear more often.
   5. A letter may appear on the board more than once.
2. When choosing to view statistics, the player may view (1) *game score statistics*, or (2) *word statistics*.
3. For game score statistics, the player will view the list of scores, in descending order by final game score, displaying:
   1. The final game score
   2. The number of times the board was reset
   3. The number of words entered in the game

The player may select any of the game scores from this list to view the settings for that game’s *board size, number of minutes,* and the *highest scoring word* played in the game (if multiple words score an equal number of points, the first played will be displayed)*.*

1. For the word statistics, the player will view the list of words used, starting from the most frequently played, displaying:
   * 1. The word
     2. The number of times the word has been played, across all games
2. The user interface must be intuitive and responsive.
3. The performance of the game should be such that students do not experience any considerable lag between their actions and the response of the application.
4. For simplicity, you may assume there is a *single system* running the application.

# Submission Instructions

To submit your assignment, you should do the following:

* Create a directory called Assignment5 in the usual **personal GitHub repository we assigned to you**. This is an **individual assignment**; do **not** use your new team repositories.
* Save your UML class diagram in the Assignment5 directory as a PDF file named “design.pdf”. **Important:** Make sure to open your PDF after generating it and double check it, as we have had a number of cases of students not realizing that the conversion to PDF had not worked as expected.
* Save the “design information” document in the same directory, in markdown format, and name it “**design-information.md**”.
* Commit and push your file(s) to your remote repository.
* Submit the commit ID for your solution on Canvas.