Assignment 5: Software Design

To continue a step further than the cipher you implemented in Assignment 4, in this assignment you will design a cryptogram game which uses a simple substitution cipher. 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 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). 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:

…

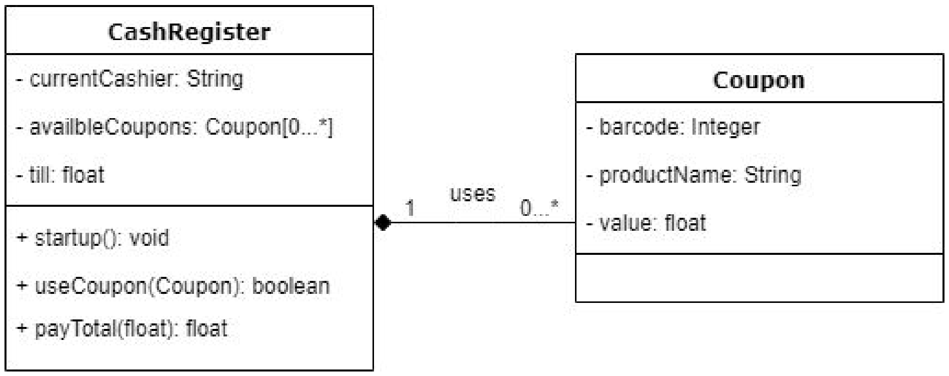
2. After starting the cash register, the cashier enters her name, and the total amount of money available in the till.

```
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. A user will be able to choose to log in as a (new or existing) *player* or as the *administrator* when starting the application. For simplicity, any authentication is optional (so you do not need to consider passwords and related security), and you may assume there is a *single system* running the application.
2. The application will allow existing *players* to (1) create a cryptogram, (2) solve a random cryptogram, and (3) view the *list of player scores*.
3. The application will allow the *administrator* to (1) view the *list of cryptogram statistics*, and (2) disable a *cryptogram.*
4. A cryptogram will have a solution (the plaintext phrase), an encoded phrase, a title and a hint phrase.
5. When a *new* player logs in, they will enter the following player information:
    a. A unique username

    b. An email address

A new player will start with 20 points in their total number of points.

6. To add a new cryptogram, a player will:
    a. Enter a unique cryptogram title.
    b. Enter a solution (unencoded) phrase.
    c. Select unique encoded letters to replace each unique letter in the solution phrase. No letter may be encoded to itself.
    d. View the resulting encoded phrase, with capitalization and any non-alphabetic characters preserved.
    e. Enter a hint phrase (to help players solve the puzzle).
    f. Edit the information entered by repeating steps a through d as necessary.
    g. Save the complete cryptogram.
    h. View a confirmation that the name assigned to the cryptogram is unique and return to the main menu, or be returned to editing the cryptogram after any error is displayed. The player's total number of points will be increased by 5 for creating a new cryptogram.
7. When a player chooses to solve a random cryptogram, they will:
    a. Be prompted to bet a number of their points on solving the next cryptogram, with a minimum of 1 and a maximum of 10, or all of their points, whichever is less.
    b. They will then be shown a random cryptogram which they have not attempted to solve before and did not create. The system will display the cryptogram title and encrypted text along with a number of *attempts* remaining in the game, starting at *five*.
    c. The player will match possible replacement and encrypted letters together, and view the resulting potential solution.
    d. When all letters in the cryptogram are replaced and they are satisfied with the potential solution, the player may submit their answer.
    e. The player will view a result indicating that the solution was successful, or decrementing the number of solution attempts remaining if it was unsuccessful.
    f. When there are *two* attempts remaining, the cryptogram *hint* will be displayed.
    g. At any point, the player may return to the menu, but choosing to solve a cryptogram will return to the same attempt for the current cryptogram.
    h. If the number of solution attempts reaches zero, they will get a result displaying that the cryptogram game was lost and the change in their total points. The number of points bet will be removed from their total. They will then return to the menu.
    i. If the player successfully solves the cryptogram, they will get a result displaying that the cryptogram game was won and the change in their total points. If the player solved the cryptogram before the hint was displayed, they will have the number of points bet added to their score. Otherwise, no points will be added and their total will be unchanged. They will then return to the menu.

8. The list of *player scores* will display a list of players in descending order of total points. The entry for each player will show their username, the number of cryptograms attempted, and their total number of points.
9. The list of *cryptogram statistics* will display a list of cryptograms from newest to oldest. The entry for each cryptogram will show its title, its creator's username, the number of games using that cryptogram completed by all players, and the percentage of wins out of total completed games. The administrator may select each cryptogram to either:
    a. View its encrypted phrase, solution phrase, and hint.
    or
    b. *Disable* the cryptogram and *penalize* its creator, to discourage creation of unfair cryptograms.
10. A disabled cryptogram will not be randomly selected in the future for any players starting a game, but will not affect any previous or in progress games.
11. The administrator may penalize the creator of a cryptogram 0-10 points (or the player's total number of points, if that is less) when they disable the cryptogram.
12. The user interface must be intuitive and responsive.
13. 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.

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 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.