**Assignment 5: Software Design**

To continue a step further than the caesar cipher you implemented in Assignment 4, in this assignment you will design a [cryptogram](http://www.cryptograms.org/faq.php) game which uses a [simple substitution cipher](https://en.wikipedia.org/wiki/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:

…

2. The application must contain a list of items in object ‘X’. Object ‘Y’ must use the list for action ‘A’.

To realize this requirement, I added to the design a class ‘X’ with list ‘Z’ of items. Class ‘Y’ has a relationship to class ‘X’ and uses it in method ‘A’ to….

...

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.

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 is frequently already some discussion and recommendations for that on Piazza.

# Requirements

1. A user will be able to choose to log in as a specific *player* or log in as the *administrator* when starting the application. For simplicity, any authentication is optional, and you may assume there is a *single system* running the application.
2. The application will allow *players* to (1) choose a cryptogram to solve, (2) solve cryptograms, and (3) view the *list of player statistics*.
3. The application will allow the *administrator* to (1) create a cryptogram, (2) create a *player*, and (3) view the *list of player statistics.*
4. A cryptogram will have a solution (the plaintext phrase) and a maximum number of allowed solution attempts for each of *three difficulty categories*.
5. To add a player, the administrator will enter the following player information:
	1. A first name
	2. A last name
	3. A unique username
	4. A difficulty category: *easy*, *normal* or *hard*.
6. To add a new cryptogram, the administrator will:
	1. Enter a unique cryptogram name.
	2. Enter a solution (unencoded) phrase.
	3. Enter the number of allowed incorrect solution attempts for the *easy* difficulty.
	4. Enter the number of allowed incorrect solution attempts for the *normal* difficulty.
	5. Enter the number of allowed incorrect solution attempts for the *hard* difficulty.
	6. Edit any of the above steps as necessary.
	7. Save the complete cryptogram.
	8. 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.
7. The encrypted phrase for the cryptogram will be generated for each player starting a new cryptogram by:
	1. Replacing each letter with another letter randomly, so that all of any particular letter are replaced with the same other letter, such as all A’s becoming C’s, and every letter is paired with a unique encrypted letter.
	2. Preserving the capitalization in the original phrase.
	3. Preserving any non-alphabetic characters (such as punctuation or white space) unaltered.
8. To choose and then solve a cryptogram, a player will:
	1. View the list of all unsolved cryptograms alongside their status as *in progress* or *unstarted*, and choose a cryptogram to solve.
	2. View the chosen cryptogram and number of incorrect solution attempts remaining (starting at whatever number is allowed for the player’s difficulty level for that cryptogram). If the cryptogram has not been played by this player before, the fully encrypted phrase should be generated and displayed. If the cryptogram is in progress, the previous state of the phrase should be displayed.
	3. Match the replacement and encrypted letters together, and view the resulting potential solution.
	4. When all letters in the cryptogram are replaced and they are satisfied with the potential solution, submit their answer.
	5. Get a result indicating that the solution was successful, or decrementing the number of incorrect solution attempts remaining if it was unsuccessful.
	6. At any point, the player may return to the list of unsolved cryptograms to try another.
	7. If the number of incorrect solution attempts reaches zero, they will get a result that the cryptogram game was lost, and this cryptogram will be marked as complete, unavailable for this player to attempt again. They will then return to the menu.
	8. If the player successfully solves the cryptogram, they will get a result that the cryptogram game was won, and this cryptogram will be marked as complete, unavailable for this player to attempt again. They will then return to the menu.
9. The list of *player statistics* will display a list of players in descending order of number of cryptograms won. The entry for each player will show their first name, the number of cryptograms won, and the number of cryptograms lost. An administrator should also see the username and difficulty status of the player.
10. The user interface must be intuitive and responsive.
11. 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.