ENGR 101 - Introduction to Programming

Mini Project 3

December 18, 2018

(Due on January 2, 2019 by 5 pm)

In Mini Project 2, you have developed a console-based Trivia quiz application by employing *procedural programming* (i.e., a set of functions and function calls). In this mini project, you are going to re-implement the same console-based Trivia quiz application by using object-oriented programming practices! The interface from user's perspective will stay the same. However, you will have a significantly different implementation in the source code level. The primary goal of this mini project is to practice using classes and objects. Hence, please use every opportunity to employ object-oriented programming concepts in your code. **At minimum**, you should create and use the following classes implemented with the below-provided list of **attributes** and **methods**. If you see a need, you *may add* additional classes and/or methods/attributes that are not in the following list.

List of Classes with their Attributes and Methods

- **User**
  - Attributes:
    - name
    - balance
    - phone
    - is_disqualified
      - False by default, and it will be set to True after the user is eliminated from the game
  - Methods:
    - print_user_stats( )
      - It should print the name, balance, phone number of the user.
- **Answer**
  - Attributes:
    - text
    - answer_no
      - Represents the number that will be printed next to the answer when it is displayed under a question.
    - is_correct
      - Will be set to True for the correct answer, and False for the others.
    - num_answering_users
      - You should store the number of users who have chosen this answer using this attribute as a counter.
  - Methods:
    - display(display_correctness, display_num_answering_users)
      - This method will print an answer in different details. display_correctness will be used to decide whether there will be any note next to the answer regarding if it is the correct answer.

display_num_answering_users will be used whether the number of users who have chosen this answer during the game will be shown.

- **Question**
  - Attributes:
    - question_text
    - answers
      - list of Answer objects: you should represent each answer of a question as an Answer object, and store them in this attribute.
    - correct_ans
      - This will keep track of the correct answer for the question. Initially, it will be None, but when answers are being added, it will be assigned to the Answer object that represents the correct answer.
  - Methods:
    - add_answer(answer_text, is_correct)
      - It will create an Answer object out of the given input values, and add this object to the answers attribute. If this is the correct answer, it should also update the value of correct_ans attribute. The answer_no for the currently added answer will be automatically set based on the current size of the answers list.
    - display(question_no, display_answers, display_correctness):
      - This method will print a question. If display_answers is True, it will also display answers of the question. If display_correctness is True, for each answer, it will also display whether that answer is correct or not.
    - process_answers (users, current_user):
      - Takes a users dictionary (see the users attribute of Game class), and a User object that represents the currently logged user who is actively playing the game now. It reads the answer of the user for the question, checks whether it is correct or not. It also randomly generates answers for the other remaining users, and checks whether their answers are correct. It then, displays whether the current user's answer is correct, and the number of users that selected each answer. It returns the number of remaining users who will continue playing the game with the next question.

- **Menu**
  - Attributes:
    - list of MenuItems
      - Each option in a menu will be represented as a MenuItem object (see the next class) and stored in a list.
    - header
      - header represents the introductory text that usually appears before the menu options, e.g., "Welcome SehirHadi Admin Section, Please choose one of the following options:"
  - Methods:
    - display(display_header)
      - Displays the menu on the screen, prompts the user for his/her selection, gets the valid user selection, and returns it to the caller. It will display the header attribute before menu items if display_header parameter is True. Otherwise, it will not show the header.

        - add_menu_item(text, number)
            - it should build and add a new MenuItem object to the list of MenuItems.

- **MenuItem**
    - Attributes:
        - text
            - Stores the text of the menu item, e.g., "Display questions for the next competition."
        - number
            - Stores the number of this menu item, e.g., 2 for the above item in admin menu, which will be used when printing the menu and getting user selection.
    - Methods:
        - display()
            - This method displays the current MenuItem object's number and text properly when called.

- **Game**
    - Attributes:
        - list of Question objects
        - dict of User objects
        - admin_menu (a Menu object)
        - prize
    - Methods:
        - play()
            - This is the central method that runs the primary logic of the game. It will call methods of other objects whenever required.
        - build_admin_menu()
            - This will be called from init(), it should build the admin menu as a Menu class object with all of its options as listed in MP2 manual.
        - show_admin_menu()
            - This method will control main flow of the admin menu when user signs in as admin "**".
        - distribute_prize()
            - This methold should distribute the prize to the winners after the competition ends, and print the results accordingly.
        - login()
            - It should manage the login process

Implementation Notes:

- Outside of class definitions, only 2 lines of code should exist that do the following:
    - In one line you will create an object of Game class.
    - In the other line, you will call a proper method on the game object that you created above to start the program.
- Please note that although __init__ methods are not specified, for every class in the above list, you should have an __init__ method in each class.

- In each method of a class, self should be the first input parameter. Self parameter is not included in the above method specifications for brevity.
- You should actively use all the methods listed under each class at least once or more. Make sure that you eliminate repeated or similar codes.

Warnings:

- **Do not** talk to your classmates on project topics when you are implementing your projects (This is serious). **Do not** show or email your code to others (This is even more serious). If you need help, talk to your TAs or the instructor, not to your classmates. If somebody asks you for help, explain them the lecture slides, but do not explain any project related topic or solution. Any similarity in your source codes will have **serious** consequences for both parties.
- Carefully read the project document, and pay special attention to sentences that involve "**should**", "**should not**", "**do not**", and other underlined/bold font statements.
- If you use code from a resource (web site, book, etc.), make sure that you reference those resource at the top of your source code file in the form of comments. You should give details of which part of your code is from what resource. Failing to do so **may result in** plagiarism investigation.
- Even if you work as a group of two students, each member of the team should know every line of the code well. Hence, it is **important** to understand all the details in your submitted code. You may be interviewed about any part of your code.

How and when do I submit my project? :

- Projects may be done individually or as a small group of two students (doing it individually is recommended for best learning experience). If you are doing it as a group, only **one** of the members should submit the project. File name will tell us group members (Please see the next item for details).
- Submit your own code in a **single** Python file. Name your code file with your and your partner's first and last names (see below for naming).
    - If your team members are Deniz Can Barış and Ahmet Çalışkan, then name your code file as deniz_can_baris_ahmet_caliskan.py (Do **not** use any Turkish characters in file name).
    - If you are doing the project alone, then name it with your name and last name similar to the above naming scheme.
    - Those who **do not** follow the above naming conventions **will get 5 pts off** of their grade.
- Submit it online on LMS by **January 2, 2019 5pm.**

Late Submission Policy:

- -10%: Submissions between 17:01 – 18:00 on the due date
- -20%: Submissions between 18:01 – midnight (00:00) on the due date

- -30%: Submissions which are up-to 24 hour late.
- -50%: Submissions which are up-to 48 hours late.
- Submission more than 48 hours late will not be accepted.

Grading Criteria? :

Code Organization (Penalty percentages over the total grade)			**Functionality**				
Not using fully meaningful variable names (Deduct up-to 20% of the total grade)	Not using or improper use of specified classes and objects (Deduct up-to 80% of the total grade)	Insufficient commenting (Deduct up-to 20% of the total grade)	Admin Menu 1 - 5 pts 2 - 15 pts 3 - 15 pts 4 - 15 pts 5 - 15 pts 6 - 5 pts	Displaying questions with their answers, and getting and evaluating current user's answer as correct or incorrect 20 pts	Simulating the other users in the background by randomly choosing answers for them, and showing stats for each answer option 20 pts	Making the game continue up until the end by properly eliminating incorrectly answering users and computing total remaining users after each question, and displaying the winners with their prizes at the end of the game 30 pts	Sign in 10 pts

Have further questions?:

Please contact your TAs if you have further questions. If you need help with anything, please use the office hours of your TAs and the instructor to get help. **Do not walk in randomly (especially on the last day) into your TAs' or the instructor's offices. Make an appointment first. This is important. Your TAs have other responsibilities. Please respect their personal schedules!**

Good Luck!