National University of Computer & Emerging Sciences, Karachi
Spring-2024, School of Computing (BSCS, BSSE, BSAI, BSCY)

Assignment # 1

Subject: Object Oriented Programming **Total Marks: 50**	**Post Date: 12th February 2024** **Due Date: 25th February 2024**
Course Instructors: Mr. Basit Ali, Ms. Sobia Iftikhar, Ms. Sumaiyah Zahid, Ms. Abeeha Sattar, Ms. Bakhtawer Abbasi, Ms. Atiya Jokhio, Mr. Minhal Raza, Ms. Rafia Shaikh, Ms. Abeer Gauher	

Instructions to be strictly followed.

1. **Each student should submit these files:**
   a. **A zip of all source files named as "A1-Q#[StudentID]" where # is the question number and Student ID is your ID.**
   b. **A DOC file where they copy code for each question and screen shot of the output. This document contains all the questions, answer codes and output in sequence. Name this document as "A1-[StudentID].docx".**
   c. **All the submissions will be made on Google Classroom.**
2. **Each output should have STUDENT ID and NAME of the student at the top.**
3. **It should be clear that your assignment would not get any credit if the assignment is submitted after the due date.**
4. **Zero grade for plagiarism (copy/ cheating) and late submissions.**

Scenario 1:

You are tasked with designing a platform named Virtual Pet Adoption System where users can adopt and care for virtual pets with advanced capabilities. The system comprises two essential classes: "Pet" and "Adopter." Your goal is to implement the system with extended features to enhance user experience and satisfaction.

Pet Class:

The Pet class represents virtual pets available for adoption. It has following features:

- healthStatus: A string indicating the health status of the pet (e.g., "Healthy," "Sick").
- hungerLevel: An integer representing the pet's hunger level.
- happinessLevel: An integer representing the pet's happiness level.
- specialSkills: A list containing special skills possessed by the pet.

Implement the following member functions within the Pet class:

- displayPetDetails(): Displays detailed information about the pet, including happiness level, health status, hunger level, and special skills.
- updateHappiness(): Updates the pet's happiness level based on user interactions.
- updateHealth(): Updates the health status of the pet, considering any changes in health.
- updateHunger(): Updates the hunger level of the pet, accounting for feeding or other relevant actions.

Moreover, if a pet is hungry their happiness also decreases by 1 and vice versa. And if you feed it the happiness increases by 1 upto max 10 happiness.

Adopter Class:

The Adopter class serves as a representation of users who are enthusiastic about adopting virtual pets. In order to enrich the functionality of this class, you are tasked with incorporating the following features: adopterName and adopterMobileNum, these attributes should be initialized during the creation of an Adopter object. A list named adoptedPetRecords within the Adopter class. This list should be responsible for maintaining detailed records of the adopted pets by the respective adopter.

Implement the following member functions within the Adopter class:

- adoptPet(): Allows the adopter to adopt a virtual pet and records its details.
- returnPet(): Enables the adopter to return a pet, updating records accordingly.
- displayAdoptedPets(): Displays detailed information about all adopted pets, including their species, happiness, health, hunger, and skills.

Create instances of the extended Pet class, showcasing diverse characteristics and skills for virtual pets. Instantiate objects of the enhanced Adopter class to represent users interested in adopting virtual pets. Demonstrate the functionalities of both classes by simulating the adoption, care, and interaction with virtual pets.

Scenario 2:

You’re bored!

You’re looking at the students going in and out of the seating at the dhaba at FAST. You decide to think of it as an OOP Scenario! You’re looking at the group of students arriving at the tables outside of the dhaba, and making mental note of how long each group of student stays at a table. For the above scenario, let’s write a program about the tables at the dhaba.

1. Each table has some properties:

- Total available seats per table (A table can only have 4 or 8 seats)
- Seats currently occupied at a table (assume only one person can occupy one seat)
- Free seats at a table
- Clean (Boolean flag representing the cleanliness of the table)

2. Each table can have some functionality associated with them:

- A default constructor – which should set the default table capacity to 4. Initially, a table will be clean and no one will be seated on it.
- A parameterized constructor – which should set the capacity to the capacity sent as parameter. If the number is not 4 or 8, it should be rounded to 4 or 8 (whichever is closest).
- Initially, a table will be clean and no one will be seated on it.
- Encapsulate the parameters of your class properly. The capacity should not be editable once it has been set by the constructor.
- A table can be used by a group of friends – In order for the table to be used, the table must first be clean. Whenever a group of friends is using the table, they will decide to use the table that can fit a group of that size. (A group of 4 will be seated at a table with 4 seats, meanwhile a group of 6 will be seated at a table with 8 seats).
- People can have lunch on the table – once the lunch is finished, the table will no longer be clean.
- People can leave the table with or without having lunch.
- Someone can clean the table – the table can only be cleaned when no one is seated at the table.

3. Create a global function called “OccupyTable” that accepts a Table array and size of the group of friends. It should find a table that is not occupied and assign a table to those people. It should mention which table has been assigned the group, and the seating capacity of the table.

4. Create a global function called “EmptyTable” that accepts a table number and sets it to empty. This should make proper changes to the variables present within that table object.

5. In your main function, you are required to perform the following actions with your Table class:

- Create an array of 5 tables.
  - Two tables should be of capacity 8, and 3 should be of capacity 4.
- Call the function OccupyTable and pass the array and 4 as its parameters. (Assume this is table 1)
- Call the function OccupyTable and pass the array and 6 as its parameters. (Assume this is table 2)

- For table 1, call the functions for:
    - Using the table
    - Having lunch on the table
    - Leaving the table
    - Cleaning the table
- Call the function EmptyTable and pass the index of table 2 as its parameter.

Scenario 3:

Assume we're writing a very bare bones program to figure out how we can apply OOP to Chess. Let's consider the following classes that will be interacting with each other to play the game.

The **ChessPiece** class is used for all chess pieces (pawn, rook, knight, etc.). Each piece has attributes such as name(King, Queen, etc.), color (black or white) and a unique symbol (K/k-for king, Q/q-for queen, N/n-for knight, etc.) to represent it on the board. Other requirements are as follows:

- Default Constructor: Whenever this constructor is called, it will create a white pawn.
- Parameterized Constructor: Creates a ChessPiece of the type specified by the parameters.
- Appropriate getters and setters for your member variables.

The **ChessBoard** class represents the chessboard itself. It contains a 2D array of **ChessPiece** to represent the 8x8 grid. Each element of the array holds a pointer to a **ChessPiece** object or is set to null if there is no piece at that position. The class has methods like **display()** to print the current state of the board, and aother method, **movePiece()**, which is responsible for moving a piece from one position to another.

More details about the functions is given below:

- Default Constructor for the ChessBoard class should initialize the 2D array of chess pieces to an initial game state (also shown below).
- The **display()** method should generate an output like this:

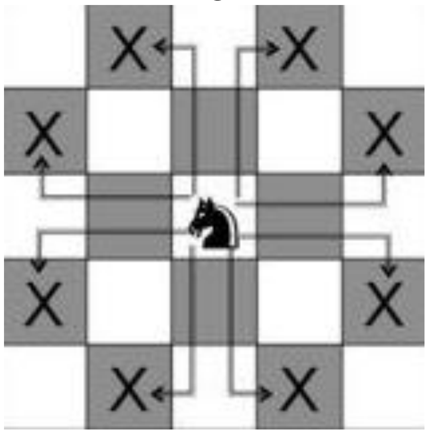
```
        a       b       c       d       e       f       g       h

8       R       N       B       Q       K       B       N       R       8

7       P       P       P       P       P       P       P       P       7

6       .       .       .       .       .       .       .       .       6

5       .       .       .       .       .       .       .       .       5

4       .       .       .       .       .       .       .       .       4

3       .       .       .       .       .       .       .       .       3

2       p       p       p       p       p       p       p       p       2

1       r       n       b       q       k       b       n       r       1

        a       b       c       d       e       f       g       h
```

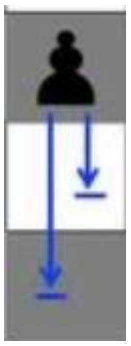
White pieces can be represented by small alphabet while black pieces can be capital alphabets.

Note: R: Rook, N: Knight, B: Bishop, Q: Queen, K: King, P: Pawn

- The **bool movePiece(string source, string destination)** method is used to move the chess piece from a source to destination. It returns true or false based on whether the move is valid or not. For simplicity's sake, let only consider the movements for knight and pawns on the first turn.
  Example: Function is called as: movePiece("b8", "a6"), so this means we are moving the knight from b8 to a6, which is a valid first move, so your function should return true. Similarly, if the function is called as: movePiece("b8", "d7"), it should return false, as d7 is already occupied by a pawn.

Notes:
The knight moves in an "L" shape on the chessboard. It can move two squares in one direction (either horizontally or vertically) and then one square in a perpendicular direction. Alternatively, it can move two squares in a perpendicular direction and then one square in the original direction.
Meanwhile, during the first move only, a pawn has two possible moves: it can move forward by one or two steps, only if there is nothing in its path.
Please see the diagrams below for further clarification.





Scenario 4:

You're being hired to write an application for different rides in a Theme Park. You're working on the Roller Coaster(woohoo!!). The Theme Park has provided you with the relevant attributes for your Roller Coaster class, and they are as follows:

- Name (of the attraction- some creative name)
- Height (maximum height that the roller coaster can reach)
- Length (total length of the roller coaster track)
- Speed (of the roller coaster)
- Capacity (amount of people that can be seated at once)
- CurrentNumRiders (number of passengers/riders currently seated in the roller coaster)
- RideInProgress (a Boolean flag, depicting whether the ride is currently in progress or not)

For the functionality, they have provided the following information:

- Constructors:
- Default – Should set the name to "roller coaster", height to 500 meters, length to 2000 meters, and capacity to 20 people. The ride should not be in progress by default.

- Parameterized – Should set the values as provided by the user. However, it should not accept a Boolean to change the ride in progress flag. It should also verify if the capacity of people is in multiples of two or three, if it is not a multiple of two or three, it should roundit to the closest multiple of two. In addition to that, the capacity should always be greater than 3.
- Appropriate Getter and Setter functions for the available variables. The same checks should be applied for the capacity variable, as applied in the parameterized constructor.
- A function to load/seat the riders into the roller coaster – Passengers/Riders can only be seated into the roller coaster if the ride is not in progress, and if there is sufficient space for all the riders.In case there is an excess number of riders compared to the available spaces, it should return the number of riders that were not seated successfully, otherwise it should return 0.
- A function to start the ride – This function can only be called if a ride is not in progress, if a ride is in progress, it should return -1. If a ride is not in progress, it needs to verify that all the seats have been occupied by the riders. In case all the seats are not occupied, it should return the number of empty seats.
- A function to stop the ride – This function can only be called if a ride is in progress. This will stop the ride.
- A function to unload the riders from the roller coaster – Passengers/Riders can only be unloaded from the roller coaster if they ride is not in progress.
- A function to accelerate the roller coaster – Every time this function is called, it should increase the speed of the roller coaster by the last non-zero digit of your roll number (If your roll number is 2034 or 2040, it should increase the speed by 4)
- A function to apply brakes to slow down the roller coaster – Every time this function is called, it should decrease the speed of the roller coaster by the first non-zero digit of your roll number. (If your roll number is 2034 or 0203, it should decrease the speed by 2)

In your main function, create two roller coaster objects by using both the constructors. Use the second object to demonstrate that your roller coaster adheres to all the conditions specified in this question.

Scenario 5:

Your task is to create a platform dedicated to connecting users with exciting BOGO (Buy One Get One) deals offered by restaurants. This platform will make it effortless for people to discover and enjoy special offers from various restaurants, allowing them to savor delicious meals with the added bonus of getting another one for free.

Restaurant Class encapsulates key details and functionalities related to each restaurant. Features include: restaurant_name, location, menu_list, price_list, valid_coupon_codes_list, and coupons_redeemed_count (static variable),a static variable tracking the total number of coupons redeemed across all instances of the Restaurant class.

Restaurant class must have following member functions:

- display_menu()
- generate_bill()
- apply_discount()

BOGOCoupon Class includes features related to coupons such as:

- coupon_code: Alphanumeric code representing the unique identity of each coupon.
- valid_from: The start date when the coupon becomes active.
- valid_until: The expiration date marking the end of the coupon's validity.
- restaurant_code: The prefix indicating the associated restaurant.

It must have the is_valid Method which validates whether the coupon is within its validity period. Also checks if the coupon is associated with the selected restaurant.

User Class must have the following attributes name, age, mobile_number, coupons_list: A list containing the BOGO coupons accumulated by the user, and redeemed_coupons_list.

It must have the following member functions:

- Accumulate_coupon()**:** Adds a new coupon to the user's list, acquired through various activities or promotions.
- Has_valid_coupon(): Checks if the user has a valid unredeemed coupon for a specific restaurant and item.
- redeem_coupon(): Validates the coupon code and ensures it hasn't been previously redeemed.

Main Details:

Two restaurants, namely Food Haven and Pixel Bites, are established with distinctive characteristics. Food Haven, located in the City Center, offers a fusion of delightful dishes such as Sushi, Pad Thai, and Mango Tango. On the other hand, Pixel Bites, situated in Cyber Street, entices users with its Digital Delicacies like Binary Burger, Quantum Quinoa, and Data Donuts.

Users are invited to explore the diverse menu offerings of Food Haven and Pixel Bites through the display_menu method. BOGO coupons are introduced with restaurant-specific codes. For instance, a coupon with the code "FH-BOGO-12345" is associated with Food Haven, and another with "PB-BOGO-67890" is linked to Pixel Bites. When placing an order, users employ the redeem_coupon process. The system validates the coupon code, ensuring it corresponds to the selected restaurant and has not been previously redeemed. Successful redemption allows users to enjoy a delightful BOGO offer on their orders, contributing to a rich and immersive dining experience.