National University of Computer & Emerging Sciences, Karachi
Fall-2023 School of Computing (BS-SE)

Assignment # 1

Subject: Object Oriented Programming **Total Marks: 50**	**Post Date: 7th September 2023** **Due Date: 22nd September 2023**
Course Instructor: Ms. Abeeha Sattar	

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.

Best of Luck! :)

Question#1: Basics + Class Diagrams [10 marks]

Suppose you're working on an application for a Smart Home Environment. You have been assigned the "Temperature Monitoring and Management" aspect of the smart home. Identify all the attributes and methods that you might need in order to monitor and manage the temperatures. (Some suggestions are mentioned in the note below, you are free to add your own attributes and functionalities as long as they are relevant)

For the above scenario:

1. List down all the properties and methods for your class (you are free to choose a class name, but it should be a reasonable name)
2. Create a class diagram for your class. Include the constructors in your diagram as well.
3. Write **JAVA** code for your class, according to your class diagram. It should implement the functionalities of monitoring and managing the temperature.
4. Make sure to encapsulate your class properly (i.e., use getters and setters!!)

Note: Monitoring includes information about current temperatures, current location of the sensor/thermostat, ability to display the temperature in Celsius or Fahrenheit, temperature history, etc. Meanwhile, management would include ways to adjust the temperatures closer to the ideal temperature desired by the user, by increasing or decreasing the temperatures.

Question # 2: Basic OOP Programming [10 marks]

Create a class called Invoice that a hardware store might use to represent an invoice for an item sold at the store. An Invoice should include four data members—a part number (type string), a part description (type string), a quantity of the item being purchased (type int) and a price per item (type int).

Your class should have a constructor that initializes the four data members. Provide a setter and a getter function for each data member. In addition, provide a member function named getInvoiceAmount that calculates the invoice amount (i.e., multiplies the quantity by the price per item), then returns the amount as an int value. If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0.

Write a test program that demonstrates class Invoice's capabilities.

Question # 3: Basic OOP Programming [10 marks]

Write a program that contains a class name Employee which contains the following private data:

members:

- employeeCode
- employeeName
- dateOfJoining

Perform the following operations

- Write a function that assigns the user defined values to these variables.
- Create 4 objects of the class Employee and pass the user defined values through a parameterized constructor.
- Write a function that asks the user to enter current date and then checks if the employee tenure is more than three years or not with employee details. Also display the number of employees that have tenure more than 3 years.

Note: Use the existing **JAVA** datatypes for date and time.

Question # 4: Detailed OOP Scenario + Programming [10 marks]

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 round it 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 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.

Question # 5: More Detailed OOP Scenario + Programming [10 marks]

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.