National University of Computer & Emerging Sciences, Karachi
Spring-2025, School of Computing (BSCS, BSSE, BSAI, BSCY)

Assignment # 2

Subject: Object Oriented Programming **Total Marks: 40**	**Post Date: 11th March 2025** **Due Date: 29th March 2025**
Course Instructors: : Ms. Bakhtawar, Ms. Sobia Iftikhar, Ms. Abeer Gauher, Ms. Nida, Mr. Basit Ali, Ms. Atiya Johkio, Ms. Sumaiyah Zahid, Mr. Minhal Raza, Ms. Abeeha Sattar, Ms. Rafia.	

Instructions to be strictly followed.

---

1. **Each student should submit these files:**
   a. **A zip of all source files named as "A2-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 "A2-[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:

We're gonna be picking up where we left off in the last assignment: **FAST's Transportation System**.

For this scenario, you are required to revisit the system, and update it according to the newer concepts that we have discussed. It should include the following concepts: Inheritance, Constructor Chaining, Polymorphism (static/dynamic), Operator Overloading, etc.

Things to consider:

Introduce new classes, such as teacher and staff members who can also avail the transport. Is there a parent class that can encompass some of the functionalities for students, teachers and staff members? Are some functionalities different between users? For example, students pay on a semester-by-semester basis, and teachers pay on a monthly basis.

Can you perform operator overloading to verify if two objects are the same or not? If so, demonstrate it in your classes. For example; are two routes the same?

Feel free to refer to the question # 4 from the previous assignment for functionalities that you have used.

Now, keeping all these things in mind, show what your updated class diagram looks like (show inheritance, aggregation and composition clearly). And update your previous code accordingly.

Scenario # 2:

Let's say you are working to simulate a Haunted House based game!

Your game allows users to create multiple Haunted Houses with ghosts of their own choices. After creating a Haunted House, the user can run a simulation to see how well the Haunted House is doing in terms or visitor experience and scares!

Let's have a look at the entities involved:

- HauntedHouse – has a name and multiple ghosts (ensure runtime polymorphism here).
- Ghost – the super class for all types of ghosts. It will contain everything that is shared between different types of ghosts. For example: name of the worker playing the ghost, the scare level of the ghost (RNG 1-10), a function to perform the haunting, etc. Ghosts have some operator overloading done on the << operator to display information about them, and + operator to upgrade a ghost (results in a ghost with two people playing a bigger ghost and combined scare level of both ghosts)
- There are different ghosts who perform hauntings differently; Poltergeists (move objects), Banshees (scream loudly), and ShadowGhosts (whisper creepily).
- There are also some hybrid ghosts that may appear; for example, ShadowPoltergeist (is a shadow ghost and Poltergeist). This ghost performs both actions of a ShadowGhost and a Poltergeist.
- Visitor – a class of objects that can "visit" a Haunted house. Each visitor has a name and some amount of bravery (1-4: Cowardly, 5-7: Average, 8-10: Fearless). When a visitor visits a Haunted House, the ghosts will try to haunt them. If the scare level of the ghost is less than the bravery range of the visitor, then the visitor might laugh or taunt the ghost. If it is greater, then the visitor might scream or run away. If it's around average, then they might get a shaky voice. (please make sure to check if the scare level lies within the bravery range; DO NOT compare the numbers directly)
- Main method should make 2-3 Haunted Houses with multiple ghosts of each type.
- Main method should also have an array of Visitors (a group of friends visiting the haunted house) Try to ensure that there's at least one of each bravery level among the group.
- "Visit" is a global friend function that accepts the Visitors array and a Haunted House, and then simulates the visit.
- Do the same for each Haunted house that you made.

Scenario # 3:

Ramzan Box Delivery System (**Hints:** Inheritance & Constructor Chaining, Function Overloading, Function Overriding & Polymorphism, Friend Function & Operator Overloading, Conflict Resolution via Friend Functions.)

In the year 2030, a futuristic AI-driven delivery company, RamzanBox, specializes in delivering food and essential supplies during the holy month of Ramzan. The company operates a fleet of autonomous drones, time-traveling delivery ships, and high-speed ground pods to ensure that iftar meals and sehri packages reach their destinations on time.

Each vehicle is equipped with an AI assistant that makes decisions based on package weight, urgency (Sehri/Iftar timing), and distance while ensuring efficient and timely deliveries. The system prevents scheduling conflicts and ensures that no package arrives late, especially when delivering food for fasting individuals.

System Overview

- The Vehicle class represents all delivery vehicles, including drones, time ships, and high-speed hyperloop pods.
- A RamzanTimeShip is a special vehicle that can travel through time, ensuring that historical deliveries follow strict accuracy protocols.
- A RamzanDrone is an AI-powered aerial delivery system that specializes in delivering small iftar meals at high speed.
- A RamzanHyperPod is a ground-based high-speed transport system optimized for bulk deliveries.
- Each vehicle's AI assistant evaluates package priority based on urgency (Sehri or Iftar), weight, and destination.
- Vehicles operate on a shared AI network, but if two AI systems disagree on priority, a conflict resolution system determines the best course of action.
- Operators use customized control panels to issue delivery commands, but different vehicles interpret the same command based on their unique design and functionality.
- Vehicles may also compare efficiency metrics, especially when deciding which vehicle is best suited for critical food deliveries during Ramzan.

Class Structure & Functionality

1. Base Vehicle Class

All delivery vehicles derive from a common base class that provides fundamental functionality. It includes:

- A unique vehicle ID
- A method to calculate the optimal delivery route
- A function to determine estimated delivery time
- A static variable to track the total number of active deliveries

2. Specialized Vehicles

- RamzanDrone: Small, fast, and airborne, designed for iftar meal deliveries.
- RamzanTimeShip: Ensures historical accuracy when delivering food to different time periods.
- RamzanHyperPod: High-speed underground transport optimized for bulk food deliveries.

Each vehicle type handles deliveries differently, even when given the same destination.

3. AI Decision-Making System

Each vehicle follows a different movement strategy based on its mode of operation:

- A RamzanDrone calculates an aerial route for high-speed delivery.
- A RamzanTimeShip verifies historical consistency before proceeding.
- A RamzanHyperPod navigates an underground tunnel network for efficient bulk delivery.
- All vehicles override a general movement function to ensure their unique operational logic.

4. Operator Control Panel

The RamzanBox AI system allows operators to issue delivery commands using a control panel:

command("Deliver", packageID);

command("Deliver", packageID, urgencyLevel);

- A RamzanTimeShip treats “urgent” packages as historically sensitive, requiring validation before transport.
- A RamzanDrone interprets “urgent” as activating high-speed mode to ensure an iftar meal arrives on time.

Even though the function name remains the same, different vehicle types interpret the command differently.

5. AI Conflict Resolution System

In cases where two AI systems disagree on which vehicle should handle a delivery, a neutral decision-making system determines the best option.

- A friend function allows access to private attributes to compare vehicles based on efficiency.
- The == operator is overloaded so that vehicles can be directly compared based on speed, capacity, and energy efficiency.

Scenario # 4:

A university has a Lab Management System used by undergraduate students. The system manages different types of users, such as Students, Teaching Assistants (TAs), and Professors. Students can access basic tools and complete assignments. TAs monitor students, assist in labs. Professors have full control of labs, assign projects, and manage research.

PERMISSIONS:

STUDENT = submit assignment

TA = view projects, manage_students

PROFESSOR = assign projects , full_lab_access

Create a global function that generates the hashes of the password. Use the value given to calculate the hash. [hash = 5381]

Then use the following formula to calculate the hash [hash * 33 + c] where “c” represents a single character.

Create a global function **authenticateAndPerformAction**(User* user, string action) that performs the functionality based on the User roles and their permissions.

The **User** class:

- Has basic attributes like name, ID, list of permissions, email and hashed_password.
- A parameterized constructor to set the attributes and save passwords as hashed.
- An authenticate function which authenticates based on the correct password.
- Display function to display the information.
- A function access lab that checks the permissions of the user and provides access to the lab if allowed.

The **Student** class:

- Derived from the user class
- Override the display function
- Has a list of assignments where each index represents 1 assignment. If the assignment is submitted by the student update the status to 1 or else leave it at 0.
- Has a functionality where students can be given assignments and their status can be updated after submission.

The **TA** class:

- Derived from the Student class
- Override the display function
- Has a List of Students that are assigned to it.
- Has a List of projects that they are working on with the professor (MAX = 2) and provides functionality to view the current projects or start working on a new project if the limit is not exceeded.
- Has a functionality where the permission is checked and then students can be assigned to the TA. Each TA can manage only up to 10 students. The TA cannot be assigned more than 10 at any time.

The **Professor** class:

- Derived from the user class
- Override the display function
- Has a functionality that allows professors to work with TA's on projects. At any one time each TA can only work on 2 projects.

Create objects of all the classes. Call the display function and authenticate function as needed.