Lab 4: Encapsulation, Life Cycles, and Logic

Course Code: CSE 4302
Week: 4
Topics: Encapsulation, Validation, Object Lifecycle, and Class-wide State Management
Instructors:
Md. Bakhtiar Hasan, Assistant Professor, CSE
Ishmam Tashdeed, Lecturer, CSE
Submission Deadline: 12:05 P.M., December 03, 2025

Upload and Turn In your `.cpp` files to **Lab 4** on **Google Classroom**. If you are unable to finish all the tasks, **submit the tasks you have completed within the given time** (You have the chance to complete the remaining ones individually later on and demonstrate them during the next week's lab evaluation). No late submissions are allowed.
DO NOT forget to rename the files like `XXX_T1.cpp`, `XXX_T2.cpp`, ... (Replace `XXX` with the last three digits of your student ID).
After the submission deadline, you will undergo a one-on-one evaluation by one of the course instructors. The instructor will ask you to show your submissions on Google Classroom. They may ask you to run the code locally and ask you questions about it.

Lab Tasks

0. Home Practice Task

For all lab tasks in Lab 3 (copy the code of Lab 3 to Lab 4), modify the class definition to add constructors that assign the member variables to an appropriate initial value. Write a zero-argumented constructor as well as an argumented constructor that initializes the member variable to the passed argument's value.
Add a destructor function that displays a message: "`The destructor for the object(<name of the class>) is called.`" This demonstrates that for each object, when the lifetime ends, the destructor is called.

1. Espresso Yourself

You are writing the firmware for a new smart coffee machine in the department's tea room. Since CS students (and therefore the teachers) run entirely on caffeine, the machine must be reliable. It needs to track water and bean levels carefully to ensure no one gets a "watery disappointment" instead of a strong espresso.

Class: `CoffeeMaker`

Private Members:

- An integer representing the current water level (in milliliters).
- An integer representing the current amount of beans (in grams).
- An integer representing the maximum capacity for water.

Public Members:

(Make member functions `const` *wherever appropriate.)*

- `CoffeeMaker(...)`: Accepts max capacity for water. Initializes current water and beans to zero.
- `~CoffeeMaker()`: Prints "`Shutting down system. Cleaning internal pipes.`"
- `addResources(...)`: Accepts amounts for water and beans. Validates that additions do not exceed capacity or result in negative numbers. Prints an error if limits are breached.
- `brewCup(...)`: Accepts a strength level (1-3). Deducts water by 150 milliliters and beans based on strength. The amount of beans required to make the coffee will be $(strength \times 7)$. If resources are insufficient, prints "`Refill needed for [item]`". Here, `item` will be replaced by whichever item is insufficient.
- `reportStatus(...)`: Display the amount of beans and the percentage of water remaining.

2. Knight and Day

You are developing a simple role-playing game. You need a class to represent a "Paladin" character. The Paladin has base states, but their actual attack damage is a complex calculation involving their level, weapon strength, and a "holy boost" modifier. This calculation should remain hidden from the main game loop.

Class: `PaladinHero`

Private Members:

- An integer for the character level.
- An integer for the weapon base damage.
- An integer for the mana points.
- `calculateHolyDamage(...)`: A function that returns an integer using the formula: $(Level \times 2) + (Weapon\ Damage) + (10\%\ of\ Mana\ Points)$.

Public Members:

(Make member functions `const` *wherever appropriate.)*

- `PaladinHero(...)`: Initializes level, weapon, damage, and mana.
- `~PaladinHero()`: Prints "`The hero has fallen.`"
- `attackMonster()`: Determines the holy damage and prints "`Striking the enemy for [X] damage!`" where `X` is the calculated damage. Deducts 1 unit of mana for the effort.
- `restAndRecover(...)`: Restores mana by `X` units and increases level by 1. Here, `X` is given as the input parameter.

3. It's Not Rocket Science

Elongated Muskrat has asked you to track the deployment of Farlink satellites. Every time a satellite is launched (instantiated), it adds to the constellation. When one de-orbits (destroys), it is removed from the count. We need a live count of how many satellites are currently in orbit.

Class: `FarlinkSatellite`

Private Members:

- A float for the satellite's altitude.
- A static integer to track the number of active satellites.

Public Members:

(Make member functions `const` *wherever appropriate.)*

- `FarlinkSatellite(...)`: Sets default altitude to 430.2km. Prints "`Satellite deployed. Orbit count: [X]`", where `X` is the updated total number of satellites in orbit.
- `~FarlinkSatellite()`: Prints "`Satellite de-orbited and burned up. Orbit count: [X]`", where `X` is the updated total number of satellites in orbit.
- `adjustAltitude(...)`: Ascends/descends (based on the symbol of the given value) the satellite. Validates that it doesn't go below 230.4km (re-entry zone).
- `getOrbitCount()`: Returns the current number of active satellites. The function should not require an instance; it belongs to the class rather than to individual objects.

4. Fast 10 Your Seatbelts

You are building a racing simulator. Each car has a "Max Speed" determined by its engine type, which cannot be altered after the car is built. However, the current speed changes constantly. You need a function to determine if the current car is moving faster than an opponent car.

Class: `RaceCar`

Private Members:

- An integer for the maximum speed.
- A double for the current speed.

Public Members:

(Make member functions `const` *wherever appropriate.)*

- `RaceCar(...)`: Accepts the max speed. Initializes the constant. Sets current speed to 0.
- `~RaceCar()`: Prints "`Car returned to garage.`"
- `accelerate(...)`: Increases current speed by a given amount. Ensures it does not exceed the constant max speed. In that case, it rejects the input and prints: "`Velocity limit exceeded. Please consult the laws of physics.`"
- `isFasterThan(...)`: Accepts another `RaceCar` object. Returns `true` if the current car's current speed is higher than the other car's current speed and `false` otherwise.
- `dashboard()`: Displays the current speed and maximum capability.