|  |
| --- |
| **Computer Engineering Department - ITU** |
| **CE101L: Object-Oriented Programming Lab** |

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated:** |
| **Teaching Assistant: Zain** | **Semester: Spring 2023** |
| **Lab Engineer: Rana Hamza Shakil** | **Batch: BSCE2022** |

# **Lab 7A. Inheritance, Aggregation, and Composition**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Roll number** | **Report**  **(out of 100)** | **Scaled to 10** | **Total**  **(out of 10)** |
|  |  |  |  |  |

Checked on: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Objective**

The objective of this lab is to observe the basic knowledge of programming in C++.

## **Equipment and Component**

|  |  |  |
| --- | --- | --- |
| **Component Description** | **Value** | **Quantity** |
| Computer | Available in lab | 1 |

## **Conduct of Lab**

1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

## **Theory and Background**

**Inheritance, Aggregation, and Composition**

Inheritance is a mechanism where one class (called the derived class) inherits the properties and behaviors of another class (called the base class). The derived class can add new properties and behaviors or modify the existing ones. Inheritance is typically used to model an "is-a" relationship between classes. For example, a car "is-a" a vehicle.

Aggregation is a relationship between two classes where one class (called the container) contains one or more instances of another class (called the contained). The contained class can exist independently of the container class. Aggregation is typically used to model a "has-a" relationship between classes. For example, a company "has-a" department.

Composition is a more specialized form of aggregation where the lifetime of the contained object is dependent on the lifetime of the container object. In other words, the contained object cannot exist without the container object. Composition is typically used to model a "part-of" relationship between classes. For example, a car "has-a" an engine.

**Lab Task**

**Task A [Marks: 5]**

Please follow the following steps before starting below tasks:

1. Create a separate header file (**.h file**) for each class declaration of data members and member functions

2. Create a separate source file (**.cpp file**) for the implementation of the class member functions.

3. Create **main.cpp** file for creating objects of class and other driving code.

**Task B: Inheritance vs Aggregation vs Composition [Marks: 35]**

**Step 1:**

* **Define the base class "Vehicle"**

**Attributes:** make: a string representing the make of the vehicle, model: a string representing the model of the vehicle, year: an integer representing the year the vehicle was made, color: a string representing the color of the vehicle, passengers: a list of Passenger objects representing the passengers in the vehicle, speed (int): represents the current speed of the vehicle in miles per hour, indicatorRight & indicatorLeft (bool): represents whether a vehicle has intention to move right or left?

**Behaviors:**

accelerate(): increases the vehicle's speed, brake(): decreases the vehicle's speed,  turn(direction): turns the vehicle indicator of left or right, add\_passenger(passenger): adds a Passenger object to the list of passengers in the vehicle.

* **Define the derived class "Car" that inherits from the "Vehicle" class**

**Attributes:** num\_doors: an integer representing the number of doors on the car, is\_sedan: a boolean representing whether the car is a sedan or not, light (bool): represents whether the car's headlights are on or off.

**Behaviors:** turn\_on\_lights(): turns on the car's headlights**,** turn\_off\_lights(): turns off the car's headlights

* **Define the class "Passenger" to represent a passenger that can be aggregated inside the "Vehicle" class**

**Attributes:** name: a string representing the name of the passenger, age: an integer representing the age of the passenger

**Behaviors:** showPassengerInfo(): display age and name of the passenger

* **Define the class "Engine" that is composed inside the "Vehicle" class using the composition**

**Attributes:** type: a string representing the type of engine (e.g. gasoline, diesel, electric), horsepower: an integer representing the horsepower of the engine, isStart (bool): indicating whether the engine is start or not

**Behaviors:** start(): starts the engine, stop(): stops the engine

* **Define the class "Transmission" that is aggregated inside the "Vehicle" class using aggregation**

**Attributes:** type: a string representing the type of transmission (e.g. manual, automatic), gear (string) : showing the current gear based on the transmission type

**Behaviors:** shift\_gears(gear): changes the gear of the transmission to the specified gear

**Step 2: Implement the classes**

* Overloaded stream in and out operators, constructor, parameterized constructor, copy constructor, and destructor for all classes.

**Step 3: Demonstrate the difference between Composition, Inheritance, and Aggregation**

* Write a main function that uses the classes above and demonstrates the difference between composition, inheritance, and aggregation

|  |
| --- |
| // Paste your code here |

### Assessment Rubric for Lab

**Method for assessment:**

Lab reports and instructor observation during lab sessions. Outcome assessed:

a. Ability to conduct experiments, as well as to analyze and interpret data (P) b. Ability to function on multi-disciplinary teams (A)

c. Ability to use the techniques, skills, and modern engineering tools necessary for engineering practice (P)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Performance metric** | **Task** | **CLO** | **Description** | **Max marks** | **Exceeds expectation** | **Meets expectation** | **Does not meet expectation** | **Obtained marks** |
| 1. Realization of experiment (a) | 1 | 1 | Functionality | 40 | Executes without errors excellent user prompts, good use of symbols, spacing in output. Through testing has been completed (35-40) | Executes without errors, user prompts are understandable, minimum use of symbols or spacing in output. Some testing has been completed (20-34) | Does not execute due to syntax errors, runtime errors, user prompts are misleading or non-existent. No testing has been completed (0-19) |  |
| 2. Teamwork (b) | 1 | 3 | Group Performance | 5 | Actively engages and cooperates with other group member(s) in effective manner (4-5) | Cooperates with other group member(s) in a reasonable manner but conduct can be improved (2-3) | Distracts or discourages other group members from conducting the experiment (0-1) |  |
| 3. Conducting experiment (a, c) | 1 | 1 | On Spot Changes | 10 | Able to make changes (8-10) | Partially able to make changes (5-7) | Unable to make changes (0-4) |  |
| 1 | 1 | Viva/Quiz | 10 | Answered all questions (8-10) | Few incorrect answers (5-7) | Unable to answer all questions (0-4) |  |
| 4. Laboratory safety and disciplinary rules (a) | 1 | 3 | Code commenting | 5 | Comments are added and does help the reader to understand the code (4-5) | Comments are added and does not help the reader to understand the code (2-3) | Comments are not added (0-1) |  |
| 5. Data collection (c) | 1 | 3 | Code Structure | 5 | Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap (4-5) | Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables (2-3) | Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy (0-1) |  |
| 6. Data analysis (a, c) | 1 | 4 | Algorithm | 20 | Solution is efficient, easy to understand, and maintain (15-20) | A logical solution that is easy to follow but it is not the most efficient (6-14) | A difficult and inefficient solution (0-5) |  |
| 7. Computer use (c) | 1 | 2 | Documentation & GitHub Submissions | 5 | Timely (4-5) | Late (2-3) | Not done (0-1) |  |
|  | Max Marks (total): | | | 100 | Obtained Marks (total): | | |  |

Lab Engineer Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_