Homework 4: Inheritance and Composition

CS 219 - Advanced Data Structures

October 26, 2024

Due Date: [11/10/2024 11:59 PM]

Abstract

This assignment focuses on applying object-oriented programming concepts in C++ to model real-world scenarios. You will work on two problems within this assignment that require implementing classes, inheritance, polymorphism, and proper memory management. Utilize the provided example codes from the Week 10 folder on Blackboard (`week9-3.cpp` and `week9-9.cpp`) to assist you in completing these tasks.

Contents

Instructions

1. Read each question carefully and adhere to the specified requirements.
2. Write clean, well-documented code with appropriate comments.
3. Ensure proper memory management to prevent leaks.
4. Submit your source code files (`.cpp` and `.h` (*if applicable*)) along with any necessary documentation.

1 Question 1: School and Student Management System

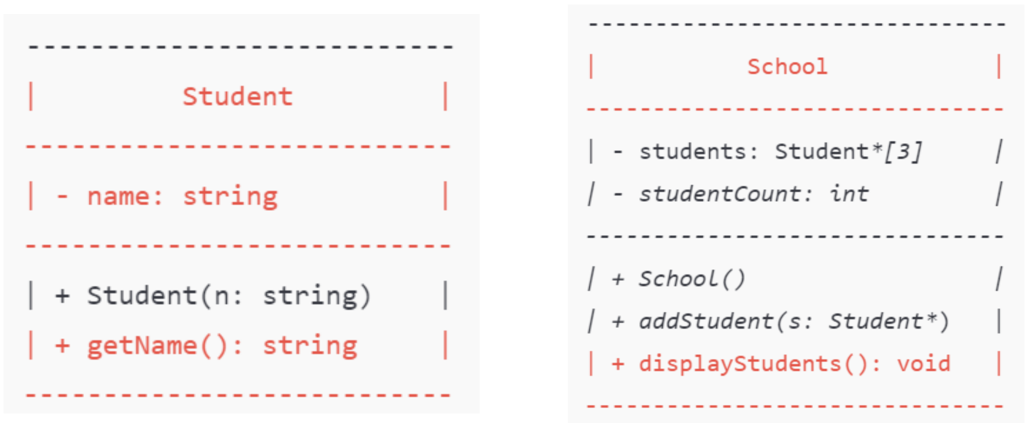
Description

Develop a simple management system for a school that handles student enrollments. The system should simulate a real-world scenario where a school can enroll a limited number of students and maintain their records.

Objectives

- Implement classes to represent a `School` and `Student`.
- Use class composition to model the relationship between a school and its students.
- Practice encapsulation and data hiding in object-oriented programming.

UML Diagram



```
---------------------------
|         Student         |
---------------------------
| - name: string          |
---------------------------
| + Student(n: string)    |
| + getName(): string     |
---------------------------
```

```
-------------------------------
|           School            |
-------------------------------
| - students: Student*[3]     |
| - studentCount: int         |
-------------------------------
| + School()                  |
| + addStudent(s: Student*)   |
| + displayStudents(): void   |
-------------------------------
```

Figure 1: UML Diagram for Question 1

Requirements

1. **Student Class**
   - **Attributes** (*private*):

    - `name`: A string representing the student's name.
  - **Methods**:
    - **Constructor**: Initializes the student's name.
    - `getName()`: Returns the student's name.

2. **School Class**
   - **Attributes** (*private*):
     - `students`: An array or list to hold pointers to `Student` objects (maximum capacity of 3 students).
     - `studentCount`: An integer to track the number of enrolled students.
   - **Methods**:
     - **Constructor**: Initializes `studentCount` to 0.
     - `addStudent(Student* student)`: Adds a `Student` to the school if there is capacity.
     - `displayStudents()`: Displays the names of all enrolled students.

3. **Main Function**
   - Create a `School` object.
   - Create at least three `Student` objects with different names.
   - Add these students to the `School` object using `addStudent()`.
   - Display all students using `displayStudents()`.

Implementation Guidelines

- **Implement the Classes**:
  - Define the `Student` and `School` classes according to the requirements.
  - Ensure that all data members are appropriately encapsulated.
- **Handle Capacity Constraints**:
  - In `addStudent()`, check if the school has reached its maximum capacity before adding a new student.
  - Provide user feedback if a student cannot be added due to capacity limits.
- **Memory Management**:
  - If using dynamic memory allocation, ensure all allocated memory is properly released to avoid memory leaks.

Example Output

```
Student added: Alice
Student added: Bob
Student added: Charlie
Students in the school:
Alice
Bob
Charlie
```

Listing 1: Sample Output

Hint

Refer to the example code `week9-9.cpp` in the Week 10 folder on Blackboard. It covers class composition and object management, which will help you structure your classes and manage objects effectively.

2 Question 2: Vehicle Hierarchy with Inheritance and Polymorphism

Description

Design a program that models a fleet of vehicles for a transportation company. The vehicles include cars, trucks, and pickup trucks. The program should demonstrate the use of inheritance, polymorphism, and appropriate access specifiers.

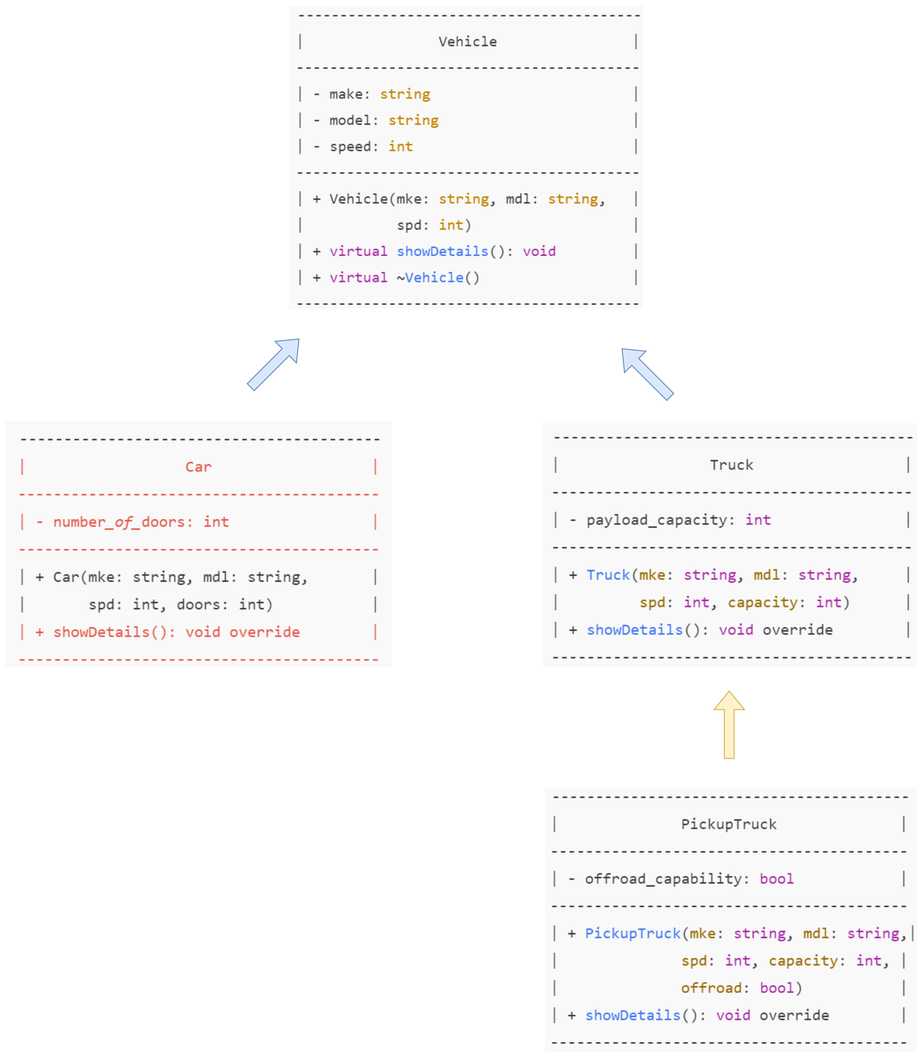
Objectives

- Implement a class hierarchy with a base class and derived classes.
- Use virtual functions to achieve polymorphism.
- Understand and apply inheritance and access specifiers.

UML Diagram

Requirements

1. **Vehicle Class (Base Class)**
   - **Attributes** (*protected*):
     - `make`: A string representing the manufacturer.
     - `model`: A string representing the model name.
     - `speed`: An integer representing the speed in km/h.
   - **Methods**:
     - **Constructor**: Initializes `make`, `model`, and `speed`.
     - `virtual void showDetails() const`: Displays the vehicle's details.
     - `virtual ~Vehicle()`: A virtual destructor for proper cleanup.
2. **Car Class (Derived from Vehicle)**
   - **Attributes** (*private*):
     - `number_of_doors`: An integer representing the number of doors.
   - **Methods**:
     - **Constructor**: Initializes attributes, including those inherited from `Vehicle`.
     - `void showDetails() const override`: Overrides `Vehicle`'s method to include car-specific details.
3. **Truck Class (Derived from Vehicle)**
   - **Attributes** (*protected*):
     - `payload_capacity`: An integer representing the payload capacity in kg.
   - **Methods**:
     - **Constructor**: Initializes attributes, including those inherited from `Vehicle`.
     - `void showDetails() const override`: Overrides `Vehicle`'s method to include truck-specific details.



```
-----------------------------------------
|                Vehicle                |
-----------------------------------------
| - make: string                        |
| - model: string                       |
| - speed: int                          |
-----------------------------------------
| + Vehicle(mke: string, mdl: string,   |
|           spd: int)                   |
| + virtual showDetails(): void         |
| + virtual ~Vehicle()                  |
-----------------------------------------
```

```
-------------------------------------------
|                   Car                   |
-------------------------------------------
| - number_of_doors: int                  |
-------------------------------------------
| + Car(mke: string, mdl: string,         |
|       spd: int, doors: int)             |
| + showDetails(): void override          |
-------------------------------------------
```

```
-------------------------------------------
|                  Truck                  |
-------------------------------------------
| - payload_capacity: int                 |
-------------------------------------------
| + Truck(mke: string, mdl: string,       |
|         spd: int, capacity: int)        |
| + showDetails(): void override          |
-------------------------------------------
```

```
-------------------------------------------
|               PickupTruck               |
-------------------------------------------
| - offroad_capability: bool              |
-------------------------------------------
| + PickupTruck(mke: string, mdl: string,|
|               spd: int, capacity: int, |
|               offroad: bool)           |
| + showDetails(): void override         |
-------------------------------------------
```

Figure 2: UML Diagram for Question 2

4. **PickupTruck Class (Derived from Truck)**
   - **Attributes** (*private*):
     – `offroad_capability`: A boolean indicating off-road capability.
   - **Methods**:
     – **Constructor**: Initializes attributes, including those inherited from `Truck`.
     – `void showDetails() const override`: Overrides `Truck`'s method to include pickup truck-specific details.

5. **Main Function**
   - Create an array or list of `Vehicle*` pointers.
   - Instantiate objects of `Car`, `Truck`, and `PickupTruck`, and store their addresses in the array.
   - Use a loop to call `showDetails()` on each `Vehicle*` pointer to demonstrate polymorphism.
   - Delete all allocated objects to prevent memory leaks.

Implementation Guidelines

- **Implement the Classes**:
  - Define the class hierarchy according to the specifications.
  - Use appropriate access specifiers (`private`, `protected`, `public`) for class members.
- **Demonstrate Polymorphism**:
  - Use virtual functions to allow derived classes to override base class methods.
  - When calling `showDetails()` through a `Vehicle*` pointer, the correct overridden method should be invoked.
- **Memory Management**:
  - Implement virtual destructors to ensure that destructors of derived classes are called correctly.
  - Delete all dynamically allocated objects at the end of the `main()` function.

Example Output

```
Car Make: Toyota, Model: Corolla, Speed: 180 km/h, Doors: 4
Truck Make: Ford, Model: F-150, Speed: 120 km/h, Payload Capacity: 1000
    kg
Pickup Truck Make: Ram, Model: 1500, Speed: 140 km/h, Payload Capacity:
    1200 kg, Offroad Capable: Yes
```

Listing 2: Sample Output

Hint

Refer to the example code `week9-3.cpp` in the Week 10 folder on Blackboard. It provides insights into inheritance and polymorphism, which are essential for this question.

Additional Notes

- **Utilize Provided Resources**:
  - The example codes in the Week 10 folder (*i.e.,* `week9-3.cpp` and `week9-9.cpp`) are valuable resources. They cover topics directly related to these questions and can help you understand how to implement similar functionalities.
- **Focus on Object-Oriented Principles**:
  - Pay attention to encapsulation, inheritance, and polymorphism.
  - Ensure your classes are well-structured and follow good programming practices.
- **Testing and Debugging**:
  - Test your programs thoroughly to ensure they behave as expected.
  - Check for memory leaks, especially if you're using dynamic memory allocation.

Submission Guidelines

1. Submit your source code files (`.cpp` and `.h` (*if applicable*)) and any additional documentation (e.g., README file) via Blackboard before the due date.
2. Ensure that your code compiles and runs correctly on the standard lab environment or specified compiler.
3. Include comments in your code to explain the functionality and logic used.

Academic Integrity

Ensure that all work submitted is your own. Plagiarism or any form of academic dishonesty will result in disciplinary action as per university policies.

Good luck, and happy coding!