UML Class Diagrams: Inheritance, Composition, and Aggregation

1. Introduction

UML class diagrams model the static structure of object-oriented systems, showing classes and relationships like inheritance, composition, and aggregation. They are vital for software design.

Objective: The purpose of this assignment is to demonstrate a comprehensive understanding of UML class diagrams by designing and illustrating the five types of inheritance **(single, multiple, multilevel, hierarchical, and hybrid), composition, and aggregation**, and to implement these relationships in C++ code, showcasing their application in object-oriented system design.

Notation Guide:

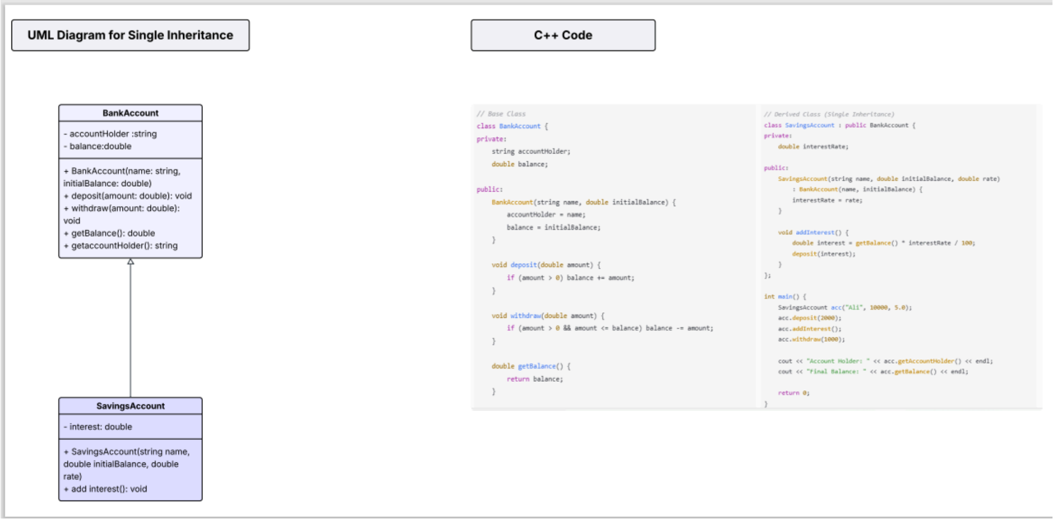
- → : Inheritance (empty head)
- ◆ : Composition (filled diamond)
- ◇ : Aggregation (empty diamond)

2. Five Types of Inheritance

Inheritance represents "is-a" relationship where subclasses inherit from parents.s

2.1 Single Inheritance

Definition: A class inherits from one parent.
Example: Class `Car` inherits from `Vehicle`.


UML Diagram for Single Inheritance
BankAccount
- accountHolder :string
- balance:double
+ BankAccount(name: string, initialBalance: double)
+ deposit(amount: double): void
+ withdraw(amount: double): void
+ getBalance(): double
+ getaccountHolder(): string
SavingsAccount
- interest: double
+ SavingsAccount(string name, double initialBalance, double rate)
+ add interest(): void
C++ Code

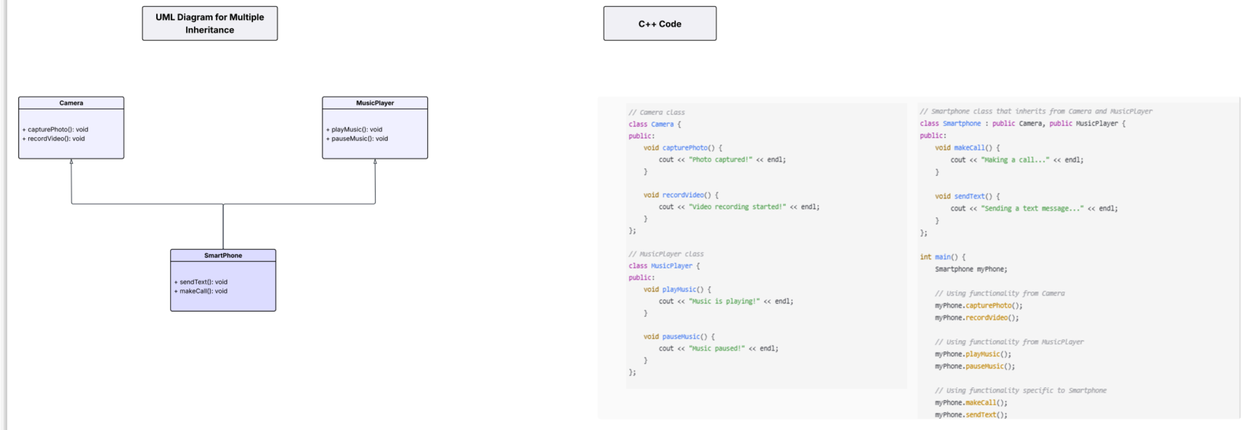
```cpp
// Base Class
class BankAccount {
private:
    string accountHolder;
    double balance;

public:
    BankAccount(string name, double initialBalance) {
        accountHolder = name;
        balance = initialBalance;
    }

    void deposit(double amount) {
        if (amount > 0) balance += amount;
    }

    void withdraw(double amount) {
        if (amount > 0 && amount <= balance) balance -= amount;
    }

    double getBalance() {
        return balance;
    }
```

```cpp
// Derived Class (Single Inheritance)
class SavingsAccount : public BankAccount {
private:
    double interestRate;

public:
    SavingsAccount(string name, double initialBalance, double rate)
        : BankAccount(name, initialBalance) {
        interestRate = rate;
    }

    void addInterest() {
        double interest = getBalance() * interestRate / 100;
        deposit(interest);
    }
};

int main() {
    SavingsAccount acc("Ali", 10000, 5.0);
    acc.deposit(2000);
    acc.addInterest();
    acc.withdraw(1000);

    cout << "Account Holder: " << acc.getAccountHolder() << endl;
    cout << "Final Balance: " << acc.getBalance() << endl;

    return 0;
}
```

Figure 1: Single Inheritance Diagram

2.2 Multiple Inheritance

Definition: A class inherits from multiple parents.
Example: Class `Smartphone` inherits from `Phone` and `Camera`.



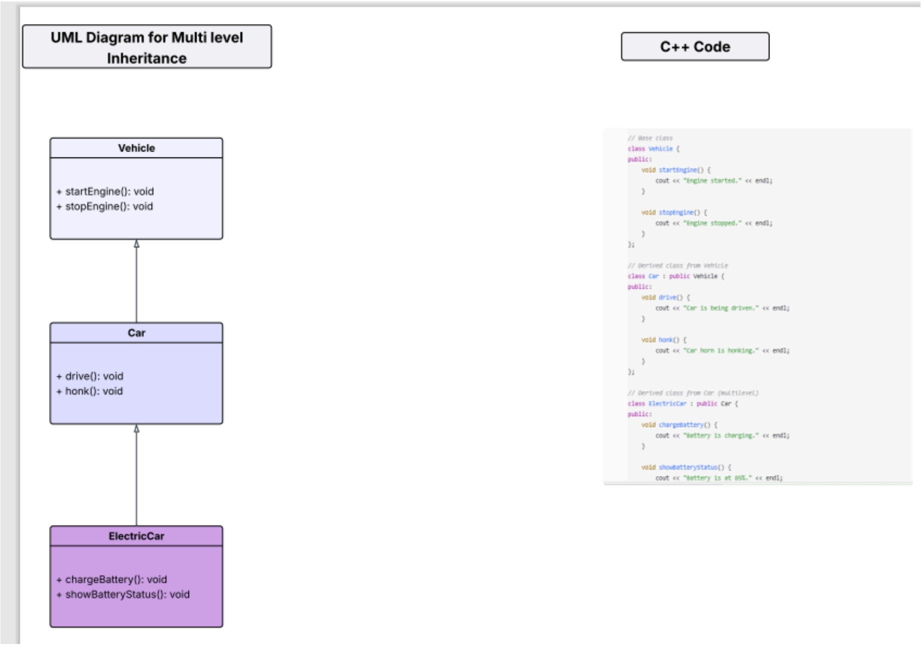
```cpp
// Camera class
class Camera {
public:
    void capturePhoto() {
        cout << "Photo captured!" << endl;
    }

    void recordVideo() {
        cout << "Video recording started!" << endl;
    }
};

// MusicPlayer class
class MusicPlayer {
public:
    void playMusic() {
        cout << "Music is playing!" << endl;
    }

    void pauseMusic() {
        cout << "Music paused!" << endl;
    }
};
```

```cpp
// Smartphone class that inherits from Camera and MusicPlayer
class Smartphone : public Camera, public MusicPlayer {
public:
    void makeCall() {
        cout << "Making a call..." << endl;
    }

    void sendText() {
        cout << "Sending a text message..." << endl;
    }
};

int main() {
    Smartphone myPhone;

    // Using functionality from Camera
    myPhone.capturePhoto();
    myPhone.recordVideo();

    // Using functionality from MusicPlayer
    myPhone.playMusic();
    myPhone.pauseMusic();

    // Using functionality specific to Smartphone
    myPhone.makeCall();
    myPhone.sendText();
```

Figure 2: Multiple Inheritance Diagram

2.3 Multilevel Inheritance

Definition: A class inherits from a parent that inherits from another.
Example: Class `SportsCar` inherits from `Car`, which inherits from `Vehicle`.

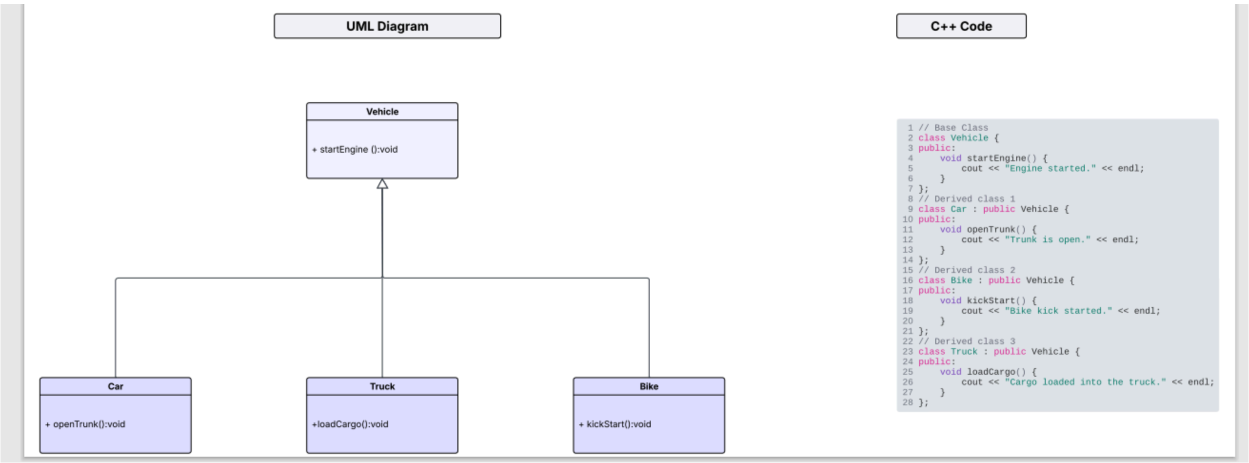


```cpp
// Base class
class Vehicle {
public:
    void startEngine() {
        cout << "Engine started." << endl;
    }

    void stopEngine() {
        cout << "Engine stopped." << endl;
    }
};

// Derived class from Vehicle
class Car : public Vehicle {
public:
    void drive() {
        cout << "Car is being driven." << endl;
    }

    void honk() {
        cout << "Car horn is honking." << endl;
    }
};

// Derived class from Car (multilevel)
class ElectricCar : public Car {
public:
    void chargeBattery() {
        cout << "Battery is charging." << endl;
    }

    void showBatteryStatus() {
        cout << "Battery is at 85%." << endl;
```

Figure 3: Multilevel Inheritance Diagram

2.4 Hierarchical Inheritance

Definition: Multiple classes inherit from one parent.
Example: Classes `Car` and `Truck` inherit from `Vehicle`.



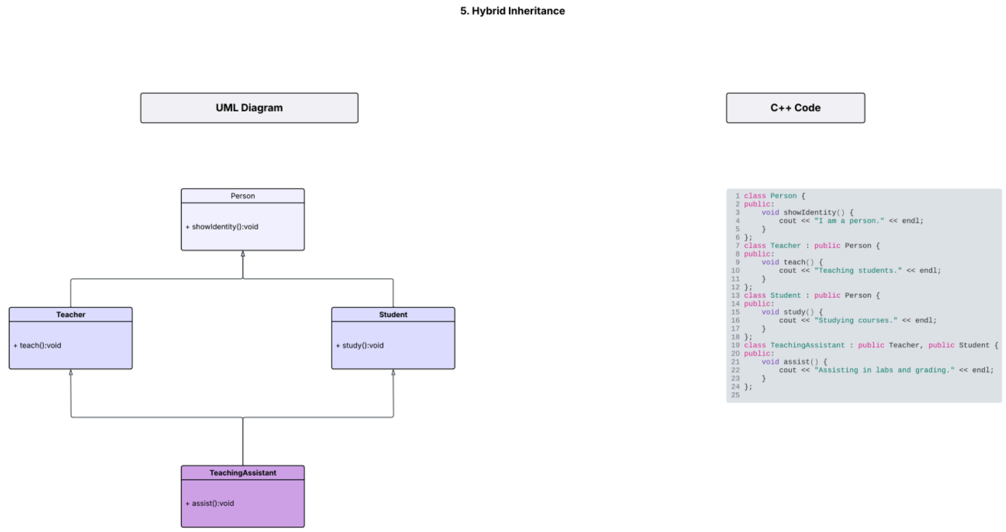
```cpp
// Base Class
class Vehicle {
public:
    void startEngine() {
        cout << "Engine started." << endl;
    }
};
// Derived class 1
class Car : public Vehicle {
public:
    void openTrunk() {
        cout << "Trunk is open." << endl;
    }
};
// Derived class 2
class Bike : public Vehicle {
public:
    void kickStart() {
        cout << "Bike kick started." << endl;
    }
};
// Derived class 3
class Truck : public Vehicle {
public:
    void loadCargo() {
        cout << "Cargo loaded into the truck." << endl;
    }
};
```

Figure 4: Hierarchical Inheritance Diagram

2.5 Hybrid Inheritance

Definition: Combines multiple and multilevel inheritance.
Example: Class `AmphibiousVehicle` inherits from `Car` and `Boat`, where `Car` inherits

from Vehicle.



```cpp
class Person {
public:
    void showIdentity() {
        cout << "I am a person." << endl;
    }
};
class Teacher : public Person {
public:
    void teach() {
        cout << "Teaching students." << endl;
    }
};
class Student : public Person {
public:
    void study() {
        cout << "Studying courses." << endl;
    }
};
class TeachingAssistant : public Teacher, public Student {
public:
    void assist() {
        cout << "Assisting in labs and grading." << endl;
    }
};
```

Figure 5: Hybrid Inheritance Diagram

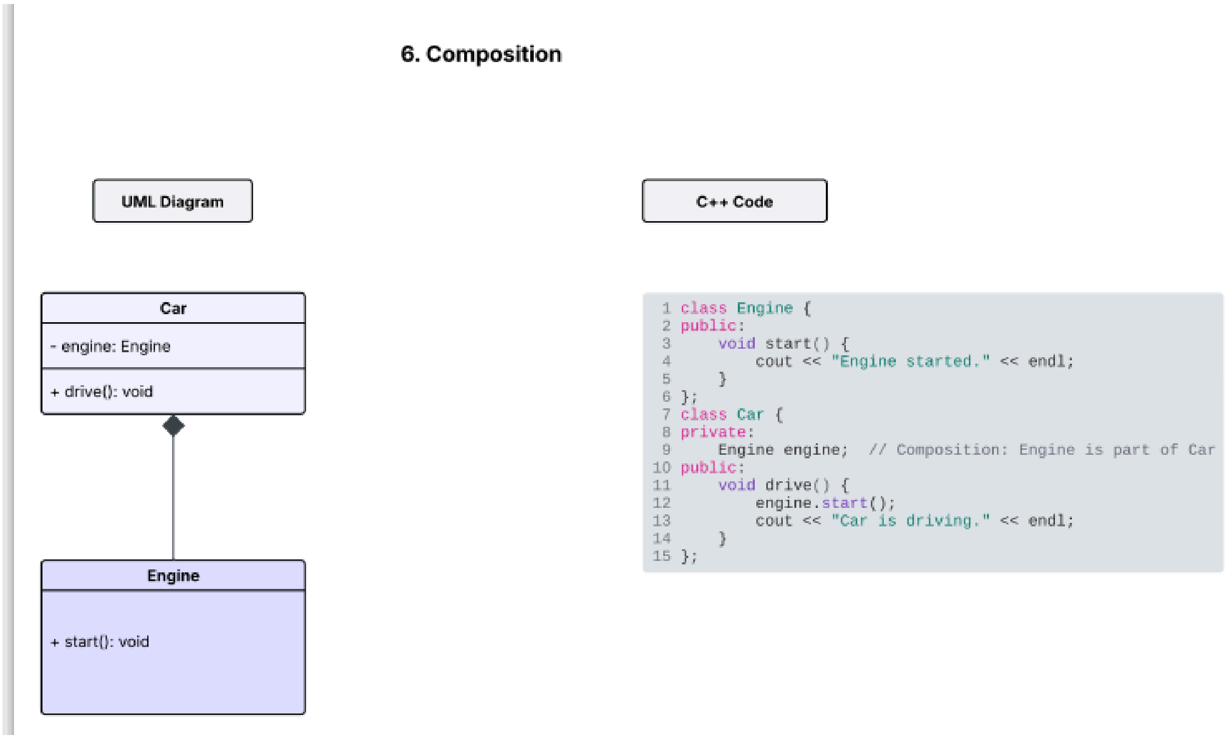
---

3. Composition and Aggregation

Composition and aggregation model "has-a" relationships, differing in ownership.

3.1 Composition

Definition: Whole-part relationship; parts cannot exist without the whole.
Example: A House contains Rooms.

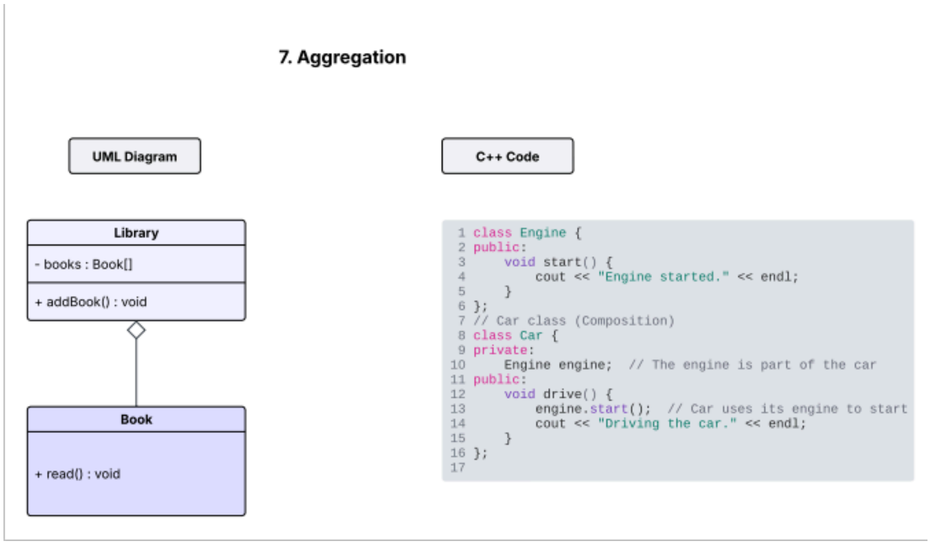


```cpp
class Engine {
public:
    void start() {
        cout << "Engine started." << endl;
    }
};
class Car {
private:
    Engine engine;  // Composition: Engine is part of Car
public:
    void drive() {
        engine.start();
        cout << "Car is driving." << endl;
    }
};
```

Figure 6: Composition Diagram

3.2 Aggregation

Definition: Whole-part relationship; parts can exist independently.
Example: A `University` contains `Students`.



```cpp
class Engine {
public:
    void start() {
        cout << "Engine started." << endl;
    }
};
// Car class (Composition)
class Car {
private:
    Engine engine;  // The engine is part of the car
public:
    void drive() {
        engine.start();  // Car uses its engine to start
        cout << "Driving the car." << endl;
    }
};

```

Figure 7: Aggregation Diagram

Comparison:

Aspect	Composition	Aggregation
Ownership	Strong	Weak
Part Lifespan	Tied to Whole	Independent

5. Key Takeaways

- UML class diagrams are essential for object-oriented design.
- Inheritance models "is-a" relationships (e.g., single, multiple).
- Composition and aggregation model "has-a" relationships.
- Practical applications include libraries, apps, and more.