Turram Toussaint

2143 OOP

Prof Griffin

**Part A: Conceptual Questions**

**Inheritance Definition** Inheritance is an object-oriented programming (OOP) concept where a derived class (child) acquires attributes and behaviors from a base class (parent). This promotes code reuse and establishes a hierarchical relationship between classes.

*Difference from Composition/Aggregation:* Inheritance forms a "is-a" relationship (e.g., a Car is a Vehicle), whereas composition and aggregation form a "has-a" relationship (e.g., a Car has an Engine). Composition tightly binds objects, while aggregation maintains a weaker association.

**Types of Inheritance**

1. **Single Inheritance:** A class inherits from only one base class. *Example:* A Car class inheriting from a Vehicle class to reuse driving functionalities.
2. **Multiple Inheritance:** A class inherits from more than one base class. *Example:* A FlyingCar inheriting from both Car and Aircraft, combining their functionalities.

**Overriding Methods**

Method overriding allows a derived class to modify a base class’s behavior by redefining a method with the same signature. This customizes inherited functionality without changing the base class.

*Why override instead of adding a new method?* Overriding maintains a consistent interface while allowing specialized behavior, ensuring polymorphism works effectively.

**Real-World Analogy** A child inherits traits from their parents (e.g., eye color, height) but may develop unique behaviors. Similarly, a Car class inherits from Vehicle but can modify how it drives.

**Part B: Minimal Coding or UML**

**Option 1: Minimal Coding**

class Vehicle {

protected:

std::string brand;

public:

Vehicle(std::string b) : brand(b) {}

virtual void drive() { std::cout << "Vehicle is driving." << std::endl; }

};

class Car : public Vehicle {

private:

int doors;

public:

Car(std::string b, int d) : Vehicle(b), doors(d) {}

void drive() override { std::cout << "Car is driving with " << doors << " doors." << std::endl; }

};

int main() {

Vehicle v("Generic");

Car c("Toyota", 4);

v.drive();

c.drive();

return 0;

}

**Option 2: UML Diagram**

*Diagram:*

+-----------------+

| Vehicle |

|---------------|

| - brand |

| + drive() |

+-----------------+

|

|

+----------------+

| Car |

|--------------|

| - doors |

| + drive() |

+----------------+

**Explanation:** The Car class inherits brand from Vehicle and overrides drive() to specify how a car drives. It also introduces doors as a unique attribute.

**Part C: Short Reflection & Discussion**

**When to Use Inheritance** *Beneficial:* When multiple classes share common behavior (e.g., different types of Vehicles inheriting from a common base class). *Overkill:* When class behaviors diverge significantly, forcing unnecessary inheritance (e.g., using Animal as a base for Car).

**Method Overriding vs. Overloading**

* Overriding: Redefining a method in a derived class (runtime polymorphism).
* Overloading: Defining multiple methods with the same name but different parameters (compile-time polymorphism).

*Why inheritance relies on overriding?* Overriding allows polymorphic behavior, letting derived classes customize behavior without changing the base class.

**Inheritance vs. Interfaces/Abstract Classes** Inheritance provides code reuse and structure, whereas interfaces define contracts without implementation. Abstract classes allow some implementation but cannot be instantiated.

**Pitfalls of Multiple Inheritance** *Problem:* Diamond problem (ambiguity when multiple base classes define the same method). *Solution:* Virtual inheritance in C++ or interface-based design in Java.

**Part D: (Optional) Research**

**Inheritance in Different Languages**

* **Java:** Supports single inheritance but allows multiple interface implementations.
* **C++:** Supports multiple inheritance but introduces complexity like the diamond problem.

**Open-Closed Principle** Inheritance aligns with the Open-Closed Principle, allowing extension without modifying base class code.

*Example:* Adding a Truck class inheriting from Vehicle without modifying Vehicle.