Homework – Inheritance in OOP

Due: 03-25-2025

Part A: Conceptual Questions

1. Inheritance Definition

- Inheritance allows a class to take on properties and behaviors from another class. The new class (called the derived or child class) automatically has access to the data and methods of the base (or parent) class and can also add or override features.

Difference from composition or aggregation:

- Composition is when one class has another class as a member (like a "has-a" relationship), while inheritance is more like an "is-a" relationship. For example, a Car is a Vehicle (inheritance), but a Car has a Engine (composition).

2. Types of Inheritance

Type 1 – Single Inheritance:

One class inherits from one base class.

Example: A Dog class inherits from Animal.

Type 2 – Multiple Inheritance:

A class inherits from more than one base class.

Example: A SmartWatch class might inherit from both Watch and FitnessTracker classes.

3. Overriding Methods

- Overriding means redefining a method in the derived class that already exists in the base class. It allows the derived class to provide its own version of the behavior.

Why override instead of just adding a new method?

- Overriding keeps the interface consistent while changing the behavior. It also supports polymorphism, which allows us to call the correct version of a method even when using a base class reference.

4. Real-World Analogy

- A child inherits traits from their parents—like eye color, hair type, or last name—but also has unique features. Similarly, in OOP, a class like Car inherits basic features from Vehicle (like the ability to drive) but can have specific behavior, like opening sunroofs.

Part B: Minimal Coding

Option 1: Minimal C++ Code Example

```
#include <iostream>
using namespace std;

class Vehicle {
public:
   string brand;

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

class Car : public Vehicle {
public:
   int doors;
```

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

int main() {
    Vehicle v;
    v.brand = "Generic";
    v.drive();

    Car c;
    c.brand = "Toyota";
    c.doors = 4;
    c.drive();

    return 0;
}
```

Explanation

- In this example, Car inherits from Vehicle, meaning it gets access to the brand attribute and drive() method. But Car overrides the drive() method to make it more specific. The base class gives us the general behavior, and the derived class makes it more specialized.

Part C: Reflection & Discussion

1. When to Use Inheritance

Good scenario:

When multiple classes share common properties and behavior — like Dog, Cat, and Bird all being derived from Animal.

Overkill scenario:
If you're just reusing one or two methods from a base class, inheritance might make things unnecessarily complex. Composition could be better for those cases.

2. Method Overriding vs. Overloading

Overriding: Redefining a method in a derived class that already exists in the base class. Happens at runtime (polymorphism).

Overloading: Having multiple methods with the same name but different parameters in the same class. Happens at compile time.

Why inheritance relies on overriding:

- Overriding allows derived classes to customize or extend behavior from the base class while still fitting into the same method interface, enabling flexibility and polymorphism.

3. Inheritance vs. Interfaces/Abstract Classes

- An abstract class can provide partial implementation, while an interface (in some languages like Java) only defines method signatures without any implementation.

Key difference:

- Inheritance gives the child class both implementation and structure. Interfaces only provide structure (i.e., what methods must exist), not how they work. Interfaces are great when you want multiple classes to share a common contract but implement things differently.

4. Pitfalls of Multiple Inheritance
Problem:

- The diamond problem — when two base classes have a method with the same name and a derived class inherits both, the compiler may not know which method to use.

Solution:

- In C++, you can use virtual inheritance to avoid duplicating shared base classes. In languages like Java, you avoid this by using interfaces instead of multiple class inheritance.