SCS1310: Object-Oriented Modelling and Programming

Composition

Viraj Welgama

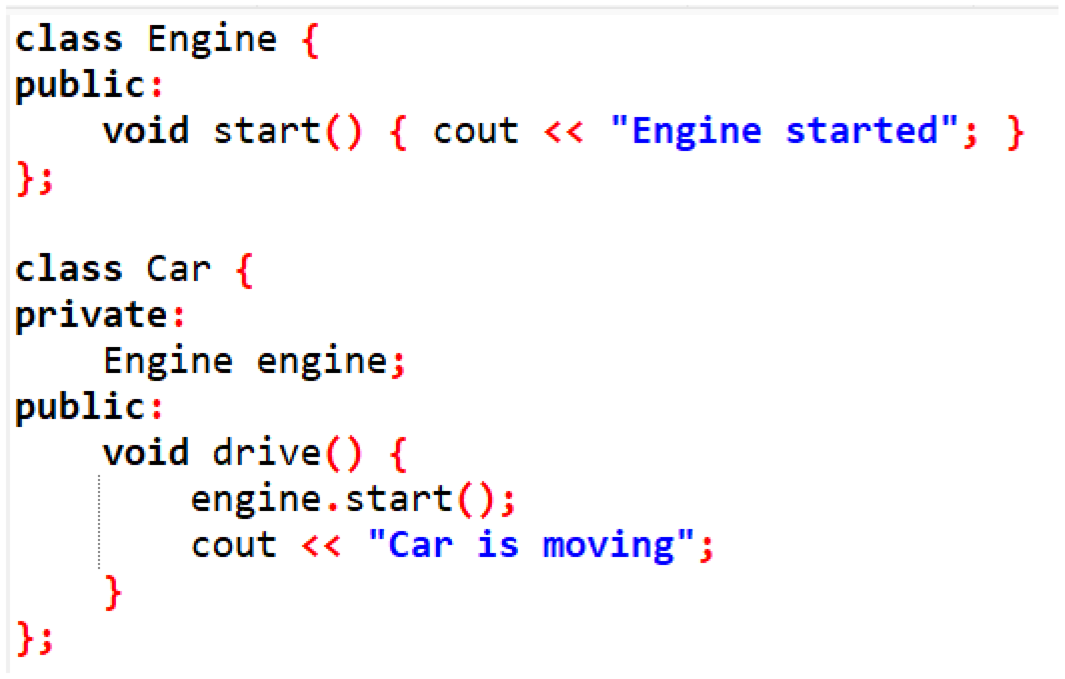
Composition

- composition is a design principle where one class contains an object (or objects) of another class as part of its attributes.
- It is a "**has-a**" relationship, meaning that one object is made up of other objects.
  - Car and Engine
  - Library and Books
  - Computer and Processor
  - Company and Employees
  - House and Rooms

Key Features of Composition

- Stronger Relationship than Inheritance
  - Unlike inheritance (which represents an "is-a" relationship), composition models a "has-a" relationship where one class is composed of one or more objects from another class.
- Encapsulation & Reusability
  - Helps keep objects modular and reusable by combining smaller components into a more complex structure.
- Dependency
  - The containing object (also called the parent or owner) is responsible for the lifecycle of the composed objects (also called child objects).

Composition: Example



```
class Engine {
public:
    void start() { cout << "Engine started"; }
};

class Car {
private:
    Engine engine;
public:
    void drive() {
        engine.start();
        cout << "Car is moving";
    }
};
```

When to Use Composition?

- When objects should contain other objects rather than extending their functionality.
- When you need more flexibility (e.g., swapping components like different types of engines in a car).
- When inheritance creates unnecessary dependencies or deep hierarchies.

Composition vs. Inheritance

Feature	Composition	Inheritance
Relationship	"Has-a"	"Is-a"
Code Reusability	High	High
Flexibility	More (as objects can be swapped)	Less (tight coupling)
Maintainability	Easier	Can lead to issues (deep hierarchy)