Turram Toussaint
2143 OOP
Prof Griffin

Composition vs. Aggregation
Composition:

Definition: Composition represents a strong "has-a" relationship where the child object is a part of the parent object and cannot exist independently. If the parent object is destroyed, the child object is also destroyed.

Example: A Car class having an Engine class as a member variable. The Engine cannot exist without the Car. If the car is destroyed, the engine is destroyed too.

```
class Engine {
   // Engine class details
};

class Car {
   private:
      Engine engine; // Engine is part of Car (Composition)
};
```

Aggregation:

Definition: Aggregation represents a weaker "has-a" relationship where the child object can exist independently of the parent object. The parent does not manage the lifecycle of the child object.

Example: A Library class that has a collection of Book objects. The Book objects exist independently and can exist without the Library.

```
class Book {
   // Book class details
};

class Library {
   private:
      Book* book; // Book can exist independently of the Library (Aggregation)
};
```

Stronger Ownership:

Composition implies a stronger form of ownership because the lifecycle of the child object is bound to the parent. If the parent object is destroyed, the child object is also destroyed.

When to Use
Composition Scenario:

Scenario: In a game, a Player class has an Inventory class that stores items. The inventory is essential for the player and doesn't exist outside the context of the player. If the player is deleted, the inventory should also be deleted. Composition is used here as the inventory's lifecycle is tied to the player.

Aggregation Scenario:

Scenario: A Student class in a school system holds a reference to Course objects. A student can take multiple courses, and a course can exist independently of the student. Aggregation works here because courses are not deleted when a student object is deleted. The course is a shared resource, and multiple students can be enrolled in it.

Differences from Inheritance
"Is-a" vs "Has-a":

Inheritance implies an "is-a" relationship, meaning a subclass is a type of the parent class. For example, a Dog is a Animal, meaning Dog inherits from Animal.

Composition and Aggregation imply a "has-a" relationship, where one object contains another. For example, a Car has an Engine (composition), or a Library has Books (aggregation).

Composition over Inheritance:

Why favor composition: Composition is preferred when flexibility and loose coupling are needed. It allows for more reusable code, where objects can be easily swapped in and out. Inheritance can lead to tight coupling and rigidity.

Real-World Analogy
Composition: A car has an engine (strong ownership). The engine cannot exist independently outside the car.

Aggregation: A car can have a driver (looser ownership). The driver can exist independently of the car, and the car does not own the driver.

Why it matters in code: Understanding these distinctions ensures proper object lifecycle management and prevents unnecessary dependencies, leading to cleaner and more maintainable code.

Part B: Minimal Class Design or UML
Option 1: Minimal Class Example

```
class Address {
   private:
      string street;
      string city;
   public:
      Address(string street, string city) : street(street), city(city) {}
};

class Person {
```

```
    private:
        string name;
        Address* address;  // Aggregation: Address can exist independently of Person
    public:
        Person(string name, Address* address) : name(name), address(address) {}
};
```

In this example, the Person class holds a reference to an Address, but the Address can exist independently. This is an aggregation relationship because Address is not owned by Person. If Person is deleted, the Address still exists.

Option 2: UML Diagram
For the UML diagram, I'll describe it in textual form:

Classes: Person and Address.

Relationship: Aggregation (open diamond arrow).

Multiplicity: Person has one Address, and each Address can be shared among multiple Person objects (1..1 for Person to Address, 0..* for Address to Person).

Part C: Reflection & Short Discussion
Ownership & Lifecycle
Composition: When a parent object (containing a child object in a composition relationship) is destroyed, the child object is also destroyed.

Aggregation: The child object can continue to exist even after the parent object is destroyed because the parent does not manage the lifecycle of the child.

Advantages & Pitfalls
Advantage of Composition: It allows for better control over object lifecycles, ensuring that when a parent is deleted, all its dependent components are also deleted, preventing memory leaks.

Pitfall of Composition: Using composition where aggregation is needed can lead to unnecessary dependencies. For example, if a Student class had a Course class as a composition instead of aggregation, deleting a Student would also delete the Course, which doesn't make sense in most cases.

Contrast with Inheritance
"Has-a" vs "Is-a": "Has-a" relationships (composition/aggregation) represent containment, while "Is-a" relationships (inheritance) represent type specialization.

Avoiding Inheritance: Composition and aggregation offer greater flexibility, as they avoid the tight coupling that can occur with inheritance. For example, instead of making Dog a subclass of Animal and inheriting all its behavior, a Dog could have an Animal (composition) or interact with Animal (aggregation), allowing for more flexibility in how objects are composed.