PART A

1. Composition vs. Aggregation:In your own words, define both composition and aggregation.Provide a concise example (with or without code) illustrating each concept.Which relationship implies a stronger form of ownership?

Composition: A type of association where one object owns another object, and the lifetime of the contained object is tied to the lifetime of the container object.Example :

```
class House {

    Room livingRoom;

    Room bedroom;}
```

Aggregation: A type of association where one object contains another object, but the contained object can exist independently of the container. Example:

```
class Library {

    vector<Book*> books;}
```

Composition means the contained objects depend on the container if the container is destroyed, the contained objects are destroyed too. Aggregation means the contained objects can exist independently, even if the container is destroyed.

2. When to Use:Describe a scenario (in any domain, e.g. gaming, banking, or UI) where composition is more appropriate than inheritance.Provide another scenario where aggregation is sufficient. Why is it only partial ownership or a looser coupling?

In a game, a Character can have different Weapons or Armor. These items are not part of the character's essence. If the character is destroyed, the items can still exist and be used by other characters. This is a composition because the character “has” equipment, but it's not tightly tied to its existence.

A Bank has many Customers, but a customer can exist without a specific bank. The bank doesn't “own” the customer; customers can have accounts in multiple banks. If the bank is destroyed, the customers still exist. This is aggregation because the bank only gathers customers without tightly controlling their existence.

In aggregation, the contained object (like a customer) can exist independently of the container (like a bank). The objects are not tightly connected, unlike in composition where the contained object depends on the container’s lifecycle.

3. Differences from Inheritance:Explain how composition/aggregation differ from the “is-a” relationship implied by inheritance.Why might an OOP design favor composition over inheritance in certain cases?

Inheritance is an “is-a” relationship, where one class is a type of another. For example, a Dog is a Animal.

Composition and Aggregation are “has-a” relationships, where one class has or uses another, but doesn't become that type. For example, a Car has an Engine.

Composition is more flexible because it lets you change parts of an object without affecting others. It avoids tight coupling between classes, making the system easier to maintain. Inheritance can lead to complex hierarchies, but composition lets you combine objects in simpler ways, making code easier to understand and reuse.

4. Real-World Analogy:Think of a real-life system that uses both composition and aggregation. (e.g., A car with an engine (composition) and a driver (aggregation)?).Why do these distinctions matter in code?

Composition:A Vehicle has an Engine. The Engine is an essential part of the Vehicle, and it cannot function without it. If the Vehicle is destroyed, the Engine is also destroyed because it's tightly coupled to the Vehicle. The Engine can’t exist without the Vehicle in this case.

Aggregation:A Vehicle has a Driver, but the Driver can exist independently of the Vehicle. The Driver can drive different vehicles or even not drive at all. The Driver can be associated with different vehicles over time, and if the Vehicle is destroyed, the Driver is not affected.

Composition ensures that when a parent object (like the Vehicle) is destroyed, its dependent parts (like the Engine) are also destroyed, managing resources correctly.

Aggregation allows Drivers to exist independently and be associated with multiple vehicles without being tightly coupled to a specific vehicle. This gives flexibility in how you manage the Driver object across different vehicles.

PART B

1.Option 1: Minimal Class Example

Create Two Classes:Person and Address, for instance.Decide if Address is truly contained by Person (composition) or if Person only holds a reference that could exist independently (aggregation).

Structure & Methods

Show minimal attributes (e.g., name, street, city).Provide a constructor or an assignment method.Emphasize how the lifecycle of one object depends (or does not depend) on the other.

Short Snippet

class Person {

private Address address; // Decide if it's a strong or weak

"has-a"

...}

class Address {...}

```
//Composition (Strong "has-a")
class Person {
private:
    Address address;  // Composition: Person "owns" the Address
public:
    string name;
    Person(const string& name, const string& street, const string& city)
        : name(name), address(street, city) {}
};

//Aggregation (Weak "has-a")
class Person {
private:
    Address* address;  // Aggregation: Person "has" the Address, but does not own it.
public:
    string name;
    Person(const string& name, Address* addr) : name(name), address(addr) {}
};
```

Composition (strong "has-a"): The Person owns the Address, and if the Person is destroyed, the Address is destroyed.

Aggregation (weak "has-a"): The Person holds a reference to the Address, but the Address can exist independently of the Person.

PART C

1. Ownership & Lifecycle:In a composition relationship, what happens to the “child” object if the “parent” is destroyed or deallocated?In an aggregation relationship, how might the child object continue to exist independently?

Composition:In a composition relationship, the parent object owns the child object. If the parent is destroyed, the child is also destroyed because the parent controls the child’s life.

Aggregation:In an aggregation relationship, the parent does not own the child. If the parent is destroyed, the child can still exist because it is independent and might be used by other objects.

2. Advantages & Pitfalls:Give one advantage of using composition for controlling object lifecycles.Provide one potential pitfall if you wrongly use composition where looser coupling (aggregation) is needed.

Advantage of Composition:If a Vehicle class owns an Engine object (composition), the Engine is automatically destroyed when the Vehicle is destroyed, ensuring proper resource management.

Pitfall of Wrongly Using Composition:If a Vehicle class owns a Wheel object using composition, it can't be reused elsewhere, causing unnecessary tight coupling.

3. Contrast with Inheritance:Summarize how “has-a” relationships differ from “is-a” relationships in practical code design.?

“Has-a”: One object contains or owns another. For example, a Car has-a Wheel (composition/aggregation).

“Is-a”: One object is a type of another. For example, a Dog is-a Animal (inheritance).

Why might we avoid inheritance in situations that can be solved by composition or aggregation?

Tight coupling: Inheritance makes classes too dependent on each other, which can make the code harder to change.

Less flexibility: Composition/aggregation allows objects to be reused and modified easily, while inheritance can limit flexibility.