Composition and Aggregation in C++

Composition and aggregation are both forms of object-oriented **"has-a" relationships** used to model complex objects by combining other objects. The key difference lies in the **ownership and lifecycle** of the included objects.

---

1. Composition

- **Definition**: A strong "has-a" relationship where the contained object (part) is owned by the container (whole). The lifetime of the contained object depends on the container; if the container is destroyed, the contained object is also destroyed.
- **Implementation**: The contained object is typically a member of the container class.

Example:

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

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

class Car {
private:
    Engine engine; // Composition: Car owns Engine
public:
    void startCar() {
        engine.start();
        cout << "Car started." << endl;
    }
};

int main() {
    Car myCar;
    myCar.startCar(); // Output: Engine started. Car started.
    return 0;
}
```

- **Key Features**:
    - Contained objects are **created and destroyed with the container**.
    - Indicates **strong ownership**.

---

2. Aggregation

- **Definition**: A weak "has-a" relationship where the container uses or references the contained object but does not own it. The lifetime of the contained object is independent of the container.
- **Implementation**: The contained object is typically referenced using pointers or references.

Example:

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

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

class Car {
private:
    Engine* engine; // Aggregation: Car uses Engine
public:
    Car(Engine* eng) : engine(eng) {}
    void startCar() {
        engine->start();
        cout << "Car started." << endl;
    }
};

int main() {
    Engine myEngine;
    Car myCar(&myEngine); // Car is associated with an existing Engine
    myCar.startCar();     // Output: Engine started. Car started.
    return 0;
}
```

- **Key Features**:
    - Contained objects **exist independently** of the container.
    - Indicates **shared or no ownership**. Both **composition** and **aggregation** are used to establish a "has-a" relationship between objects in C++, but the **key difference lies in ownership, dependency, and how objects are managed.**

---

1. Composition

Key Characteristics:

- **Strong Ownership**:
    - The container (or whole) **owns** the contained objects (or parts).
    - The contained object cannot exist without the container.

- **Lifetime Dependency**:
    - If the container object is destroyed, the contained objects are also destroyed automatically.
- **Implementation**:
    - Contained objects are created as part of the container class (e.g., as member variables).
    - Typically uses normal member objects.

Use composition when:

- The parts are an **integral and inseparable** part of the whole.
- The container should be responsible for creating and destroying its parts.
    - Example: A `Car` has an `Engine`—without the car, the engine is irrelevant.

---

2. Aggregation

Key Characteristics:

- **Weak Ownership**:
    - The container **uses** or **references** the contained object but does not own it.
    - The contained object exists **independently** of the container.
- **Lifetime Independence**:
    - If the container is destroyed, the contained object **still exists**.
    - The container only holds a reference (e.g., a pointer or reference) to the contained object.
- **Implementation**:
    - Contained objects are passed to the container, typically using pointers or references. As in above example

```
int main() {
    Engine myEngine;    // Engine exists independently
    Car myCar(&myEngine); // Car uses an external Engine
    myCar.startCar();    // Output: Engine started. Car started.
    // Even if myCar is destroyed, myEngine remains.
    return 0;
}
```

Use aggregation when:

- The parts can exist **independently** of the whole.
- The container should not control the lifecycle of the parts.
    - Example: A `Team` uses `Players`, but players can exist without the team.

Key Differences

Feature	Composition	Aggregation
Ownership	Strong: The container owns the parts.	Weak: The container uses the parts.
Lifetime Dependency	Contained objects are destroyed with the container.	Contained objects outlive the container.
Implementation	Direct members (normal objects in class).	Pointers or references to objects.
Independence	Contained object cannot exist without the container.	Contained object can exist independently.
Example Relationship	A `Car` has an `Engine`.	A `Team` has `Players`.

Illustrative Analogy

Think of a **Composition** as a "body and heart" relationship:

- The body owns the heart, and if the body is destroyed, the heart is destroyed too. The heart cannot function independently.

On the other hand, an **Aggregation** is like a "university and students" relationship:

- The university uses and manages students, but students exist and live independently of the university.