Relationships in C++

Association, Aggregation, Composition and Inheritance

Relationships in C++

1. Association

- **Definition**: A general **"uses-a"** relationship between two classes.
- **Nature**: Weakest relationship. No ownership.
- **Lifetime**: Independent.
- **Example**: A Doctor works in a Hospital.

2. Aggregation

- **Definition**: A special form of association **where one class has another, but both can live independently.**
- **Nature**: "Has-a" relationship with **shared ownership**.
- **Lifetime**: Contained object can exist without container.
- **Example**: A Team has Players, but players can exist outside the team.

3. Composition

- **Definition**: A strong **"part-of"** relationship. One class **owns** the other.
- **Nature**: Strongest form of containment.
- **Lifetime**: Contained object is destroyed when the container is destroyed.
- **Example**: A Car has an Engine that can't exist without the car.

4. Inheritance

- **Definition**: A **"is-a"** relationship. A class inherits properties from another class.
- **Nature**: Code reuse and polymorphism.
- **Lifetime**: Child class depends on parent.
- **Example**: A Dog is an Animal.

Relationship	Meaning	Lifetime Dependency	Example
Association	Uses-a	No	Doctor–Hospital
Aggregation	Has-a	No	Team–Player
Composition	Part-of	Yes	Car–Engine
Inheritance	Is-a	Yes	Dog–Animal

Association

Doctor uses a **Hospital** (no ownership).

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

class Hospital {
public:
    void display() {
        cout << "Hospital services\n";
    }
};

class Doctor {
public:
    void visitHospital(Hospital &h) {
        h.display();
    }
};
```

Aggregation

Team has a **Player**, but the player exists independently (shared pointer).

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

class Player {
public:
    void show() {
        cout << "Player is ready\n";
    }
};

class Team {
    Player* player;  // Aggregation via pointer
public:
    Team(Player* p) {
        player = p;
    }

    void display() {
        player->show();
    }
};
```

Composition

Car has an **Engine** that doesn't exist without the car.

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

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

class Car {
    Engine engine;  // Composition: Engine is part of Car
public:
    void startCar() {
        engine.start();
    }
};
```

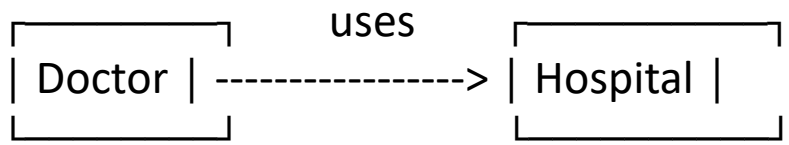
Inheritance
Dog is an **Animal**.

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

class Animal {
public:
    void speak() {
        cout << "Animal sound\n";
    }
};

class Dog : public Animal {
    // inherits speak()
};
```

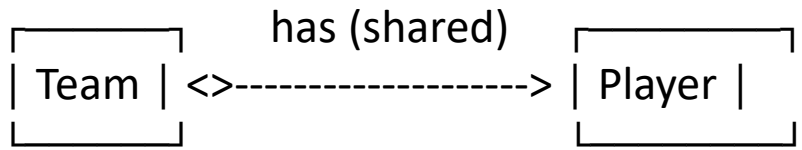
```cpp
#include <iostream>
using namespace std;

////////////////// Association //////////////////////
class Hospital {
public:
    void display() {
        cout << "[Association] Hospital services\n";
    }
};

class Doctor {
public:
    void visitHospital(Hospital &h) {
        h.display();
    }
};

/////////////// Aggregation //////////////////////
class Player {
public:
    void show() {
        cout << "[Aggregation] Player is ready\n";
    }
};

class Team {
    Player* player;  // Aggregation via pointer
public:
    Team(Player* p) {
        player = p;
    }

    void display() {
        player->show();
    }
};

/////////////// Composition /////////////////
class Engine {
public:
    void start() {
        cout << "[Composition] Engine started\n";
    }
};

class Car {
    Engine engine;  // Composition: Engine is part of Car
public:
    void startCar() {
        engine.start();
    }
};

/////////////// Inheritance ////////////////////
class Animal {
public:
    void speak() {
        cout << "[Inheritance] Animal sound\n";
    }
};

class Dog : public Animal {
    // inherits speak()
};

//////// Main Function ///////////
int main() {
    // Association
    Hospital h;
    Doctor d;
    d.visitHospital(h);

    // Aggregation
    Player p;
    Team t(&p);  // Pass pointer to existing player
    t.display();

    // Composition
    Car car;
    car.startCar();

    // Inheritance
    Dog dog;
    dog.speak();

    return 0;
}
```

[Association] Hospital services
[Aggregation] Player is ready
[Composition] Engine started
[Inheritance] Animal sound

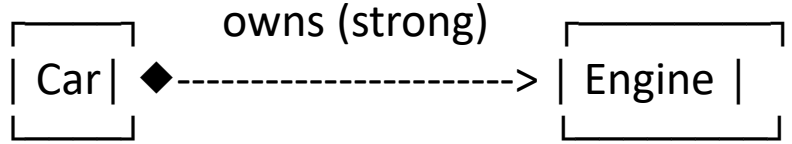
1. Association (Doctor uses Hospital)



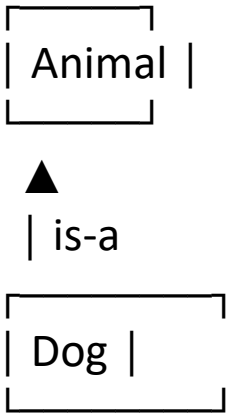
2. Aggregation (Team has Player, but Player exists independently)



3. Composition (Car has Engine, and Engine cannot exist without Car)



4. Inheritance (Dog is an Animal)



-------> : Association (uses-a)
<--------> : Aggregation (has-a, shared)
<◆--------> : Composition (part-of, strong ownership)
▲ : Inheritance (is-a)

- Association in C++ is a relationship between two classes where one class uses the functionalities provided by the other class. In other words, an association represents the connection or link between two classes. In an association, one class instance is connected to one or more instances of another class.

- Composition is a relationship between two classes in which one class, known as the composite class, contains an object of another class, known as the component class, as a member variable. The composite class owns the component class, and the component class cannot exist independently of the composite class. In other words, the lifetime of the component class is controlled by the composite class.

- Aggregation is a relationship between two classes in which one class, known as the aggregate class, contains a pointer or reference to an object of another class, known as the component class. The component class can exist independently of the aggregate class, and it can be shared by multiple aggregate classes. In other words, the lifetime of the component class is not controlled by the aggregate class.

Association

- **Usage:** Association is a basic relationship where one class is related to another class. It represents a connection between objects of different classes. For example, a **Teacher** class may be associated with a **Student** class if teachers interact with students in some way.

- **"Has a" Relationship:** In an association, you can say that **one class "has" or "uses" another class, but it's a loose relationship.** The classes are independent, and changes to one class do not necessarily impact the other class.

- Example: A Library class is associated with a Book class, where a library "has" books, but changing a book doesn't affect the library as a whole.

Association

```
class Bank {
public:
void transferMoney(Account* fromAccount, Account* toAccount, double amount)
{
// Code to transfer money from one account to another
}
};


class Account {
private:
int id;
double balance;

public:
Account (int id, double balance) : id(id), balance(balance) { }
int getId() { return id; }
double getBalance() { return balance; }

};
```

```
int main() {
Account* account1 = new Account(123, 1000.00)
Account* account2 = new Account(456, 500.00);
Bank bank;
bank.transferMoney(account1, account2, 250.00);
return 0;
}
```

Composition

Usage: Composition is a stronger form of association where one class is composed of other classes. It implies a part-whole relationship. If the whole is destroyed, the parts are also destroyed. For example, a Car class can be composed of an Engine class, Wheel classes, etc.

"Has a" Relationship: In composition, you can say that one class "has" another class, but the relationship is such that the part is a crucial component of the whole. You can't have a car without an engine.

Example: A House class is composed of Room classes. If the house is destroyed, the rooms are also destroyed.

Composition

```
class Engine {
public:
void start()
{
// Code to start the engine
}
};
```

```
class Car {
private:
Engine* engine;
public:
Car() : engine(new Engine())
{}
void startCar()
{
engine->start();
}

};
```

```
int main() {
Car car;
car.startCar();
return 0;
}
```

Aggregation

Usage: Aggregation is a weaker form of composition where one class is composed of other classes, but the parts can exist independently of the whole. It implies a part-whole relationship, but the parts are not exclusive to the whole. For example, a University class can have an aggregation relationship with a Department class. Departments can exist independently or within a university.

"Has a" Relationship: In aggregation, you can say that one class "has" another class, but the parts are not as tightly bound to the whole as in composition.

Example: A Department class is part of a University class, but departments can also exist independently.

Aggregation

```
class Person {
public:
Person(string name) : name(name), address(nullptr) {}
void setAddress(Address* address) {
this->address = address;
}
private:
string name;
Address* address;
};
```

```
int main() {
Address* address = new Address(“Street 16.", “Islamabad", “ISL", “44000");
Person person(“Ali");
person.setAddress(address);
return 0;
}
```

```
class Address {
private:
string street;
string city;
string state;
string zip;

public:
Address(string street, string city, string state, string zip) : street(street), city(city), state(state), zip(zip) {}


};
```