In C++, there are several ways to establish relationships between classes. The most common ones include:

1. Association: represents a weaker relationship between objects. It signifies that objects are related in some way, but one does not necessarily own or control the other. Objects can exist independently of each other

#include <iostream>

#include <string>

class Teacher {

public:

Teacher(const std::string& name) : name(name) {}

std::string GetName() const {

return name;

}

private:

std::string name;

};

class Student {

private:

std::string name;

Teacher myTeacher;

public:

Student(const std::string& name) : name(name) {}

std::string GetName() const {

return name;

}

void SetTeacher(const Teacher& teacher) {

myTeacher = teacher;

}

std::string GetTeacherName() const {

return myTeacher.GetName();

}

};

int main() {

Teacher mrSmith("Mr. Smith");

Student alice("Alice");

Student bob("Bob");

alice.SetTeacher(mrSmith);

std::cout << "Student: " << alice.GetName() << std::endl;

std::cout << "Teacher: " << alice.GetTeacherName() << std::endl;

std::cout << "Student: " << bob.GetName() << std::endl;

// Since Bob hasn't been associated with a teacher, this will result in an empty string.

std::cout << "Teacher: " << bob.GetTeacherName() << std::endl;

return 0;

}

2::Aggregation: Aggregation is a specialized form of association where one class "owns" or "contains" the other class, but the contained class can exist independently. It is represented by a "part-of" relationship. Here's an example:

**Aggregation::**

In C++, aggregation is a process in which one class defines another class as any entity reference. It is another way to reuse the class. It is a form of association that represents HAS-A relationship.

## **C++ Aggregation Example**

Let's see an example of aggregation where Employee class has the reference of Address class as data member. In such way, it can reuse the members of Address class.

#include <iostream>

**using** **namespace** std;

**class** Address {

**public**:

   string addressLine, city, state;

     Address(string addressLine, string city, string state)

    {

**this**->addressLine = addressLine;

**this**->city = city;

**this**->state = state;

    }

};

**class** Employee

    {

**private**:

        Address\* address;  //Employee HAS-A Address

**public**:

**int** id;

        string name;

        Employee(**int** id, string name, Address\* address)

       {

**this**->id = id;

**this**->name = name;

**this**->address = address;

       }

**void** display()

       {

           cout<<id <<" "<<name<< " "<<

             address->addressLine<< " "<< address->city<< " "<<address->state<<endl;

       }

   };

**int** main(**void**) {

    Address a1= Address("C-146, Sec-15","Noida","UP");

    Employee e1 = Employee(101,"Nakul",&a1);

            e1.display();

**return** 0;

} o/p=101 Nakul C-146, Sec-15 Noida UP

Program to call parametarized from deriver classs

#include <iostream>

using namespace std;

class Address {

public:

string addressLine, city, state;

Address(string addressLine, string city, string state)

{

this->addressLine = addressLine;

this->city = city;

this->state = state;

}

};

class Employee : public Address

{

public:

int id;

string name;

Employee(int id, string name, string a, string b,string c ): Address(a,b,c)

{

this->id = id;

this->name = name;

}

void display()

{

cout<<id <<" "<<name<< " "<<city<<" ";

}

};

int main(void) {

//Address a1= Address("C-146, Sec-15","Noida","UP");

Employee e1 = Employee(101,"Nakul","C-146, Sec-15","Noida","UP");

e1.display();

return 0;

}

3) Composition: Composition is a stronger form of aggregation where the lifetime of the contained class is controlled by the container class. It represents a "whole-part" relationship.

#include <iostream>

class Engine {

public:

void start() {

std::cout << "Engine started." << std::endl;

}

void stop() {

std::cout << "Engine stopped." << std::endl;

}

};

class Car {

private:

Engine engine;

public:

void startCar() {

engine.start();

}

void stopCar() {

engine.stop();

}

};

int main() {

Car myCar;

myCar.startCar();

myCar.stopCar();

return 0;

}

```

Composition represents a strong "whole-part" relationship between objects. In composition, the child object's existence is dependent on the parent object. When the parent object is destroyed, its child objects are also destroyed.

4) Inheritance: Inheritance represents an "is-a" relationship between classes, where one class derives from another class and inherits its properties and behaviors. Here's an example:

5) Dependency represents a relationship where one class depends on another class temporarily. It is the weakest form of relationship and often occurs when a class uses another class as a parameter or local variable.

#include <iostream>

#include <string>

// Declaration of the dependent class

class DependencyClass {

public:

void PrintMessage(const std::string& message) {

std::cout << "DependencyClass: " << message << std::endl;

}

};

// Declaration of the main class that depends on DependencyClass

class MainClass {

private:

DependencyClass dependency;

public:

void DoSomething() {

dependency.PrintMessage("Hello from MainClass!");

}

};

int main() {

MainClass mainObj;

mainObj.DoSomething();

return 0;

}

In C++ and object-oriented programming (OOP), both dependency and association describe relationships between classes, but they represent different kinds of relationships. Here's a breakdown of the difference between dependency and association:

1. Dependency:

- Dependency represents a weaker relationship between classes.

- It occurs when one class uses the functionality of another class, but the dependency is not a permanent or long-lasting relationship.

- The dependent class relies on the other class to perform a specific task or provide a certain functionality.

- The dependent class uses objects or services provided by the other class, typically through function calls or method invocations.

- The dependent class has knowledge of the other class, but there is no direct link or reference between them beyond the usage.

- Changes in the dependent class may require modifications in the other class if the functionality or interface being used changes.

- Dependency is typically represented by a parameter, local variable, or return type in method signatures.

- Dependency relationships are typically established through method calls or function invocations.

2. Association:

- Association represents a stronger and more persistent relationship between classes.

- It represents a structural connection between two classes, indicating that objects of one class are connected or related to objects of another class.

- The associated classes have a long-lasting relationship and are often aware of each other's existence.

- Association is bidirectional, meaning that the relationship can be navigated from both ends.

- It is often represented by member variables or attributes in the classes participating in the association.

- Objects of one class can directly reference objects of the other class through these member variables.

- Changes in one class may require adjustments in the other class if the association structure or behavior changes.

- Association relationships are typically established through member variables or attributes.

To summarize, dependency represents a looser and more temporary relationship between classes, where one class depends on the functionality of another class. Association, on the other hand, represents a stronger and more permanent relationship, indicating that objects of one class are associated or connected to objects of another class.