

Relationship Between Classes- Inheritance and Aggregation

Ali Haider

syedalihaider.ciit@gmail.com

Department of Computer Science IUB

Overview

- Public, Private and Protected Inheritance
- C++ Aggregation (HAS-A Relationship)
- Friend Function

Public, Private and Protected Inheritance-1

- When deriving a class from a base class, the base class may be inherited through **public**, **protected** or **private** inheritance.
- The type of inheritance is specified by the access-specifier as explained above.
- We hardly use **protected** or **private** inheritance, but **public** inheritance is commonly used.
- While using different type of inheritance, following rules are applied –
- **Public Inheritance** – When deriving a class from a **public** base class, **public** members of the base class become **public** members of the derived class and **protected** members of the base class become **protected** members of the derived class.

Public, Private and Protected Inheritance-2

- A base class's **private** members are never accessible directly from a derived class, but can be accessed through calls to the **public** and **protected** members of the base class.
- **Protected Inheritance** – When deriving from a **protected** base class, **public** and **protected** members of the base class become **protected** members of the derived class.
- **Private Inheritance** – When deriving from a **private** base class, **public** and **protected** members of the base class become **private** members of the derived class.

Example

```
class base {
    public:
        int x;
    protected:
        int y;
    private:
        int z;
};

class publicDerived: public base {
    // x is public
    // y is protected
    // z is not accessible from publicDerived
};

class protectedDerived: protected base {
    // x is protected
    // y is protected
    // z is not accessible from protectedDerived
};

class privateDerived: private base {
    // x is private
    // y is private
    // z is not accessible from privateDerived
}
```

C++ Aggregation (HAS-A Relationship)

- 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.
- Aggregation implies a relationship where the child can exist independently of the parent.
- 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.

Example of Aggregation

```
#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;
}
```

C++ Friend function

- If a function is defined as a friend function in C++, then the protected and private data of a class can be accessed using the function.
- By using the keyword friend compiler knows the given function is a friend function.
- For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword friend.
-

Declaration of friend function in C++

- `class class_name`
- `{`
- `friend data_type function_name(argument/s); // syntax of friend function.`
- `};`
- In the above declaration, the friend function is preceded by the keyword `friend`.
- The function can be defined anywhere in the program like a normal C++ function.
- The function definition does not use either the keyword **friend** or **scope resolution operator**.

Characteristics of a Friend Function

- The function is not in the scope of the class to which it has been declared as a friend.
- It cannot be called using the object as it is not in the scope of that class.
- It can be invoked like a normal function without using the object.
- It cannot access the member names directly and has to use an object name and dot membership operator with the member name.
- It can be declared either in the private or the public part.

Example

```
#include <iostream>
using namespace std;
class Box
{
    private:
        int length;
    public:
        Box(): length(0) { }
        friend int printLength(Box); //friend function
};
int printLength(Box b)
{
    b.length += 10;
    return b.length;
}
int main()
{
    Box b;
    cout<<"Length of box: "<< printLength(b)<<endl;
    return 0;
}
```

Example

```
#include <iostream>
using namespace std;
class B;           // forward declaration.
class A
{
    int x;
public:
    void setdata(int i)
    {
        x=i;
    }
    friend void min(A,B);           // friend function.
};
class B
{
    int y;
public:
    void setdata(int i)
    {
        y=i;
    }
    friend void min(A,B);           // friend function
};
```

```
void min(A a,B b)
{
    if(a.x<=b.y)
        std::cout << a.x << std::endl;
    else
        std::cout << b.y << std::endl;
}

int main()
{
    A a;
    B b;
    a.setdata(10);
    b.setdata(20);
    min(a,b);
    return 0;
}
```