OOP: Inheritance in C++

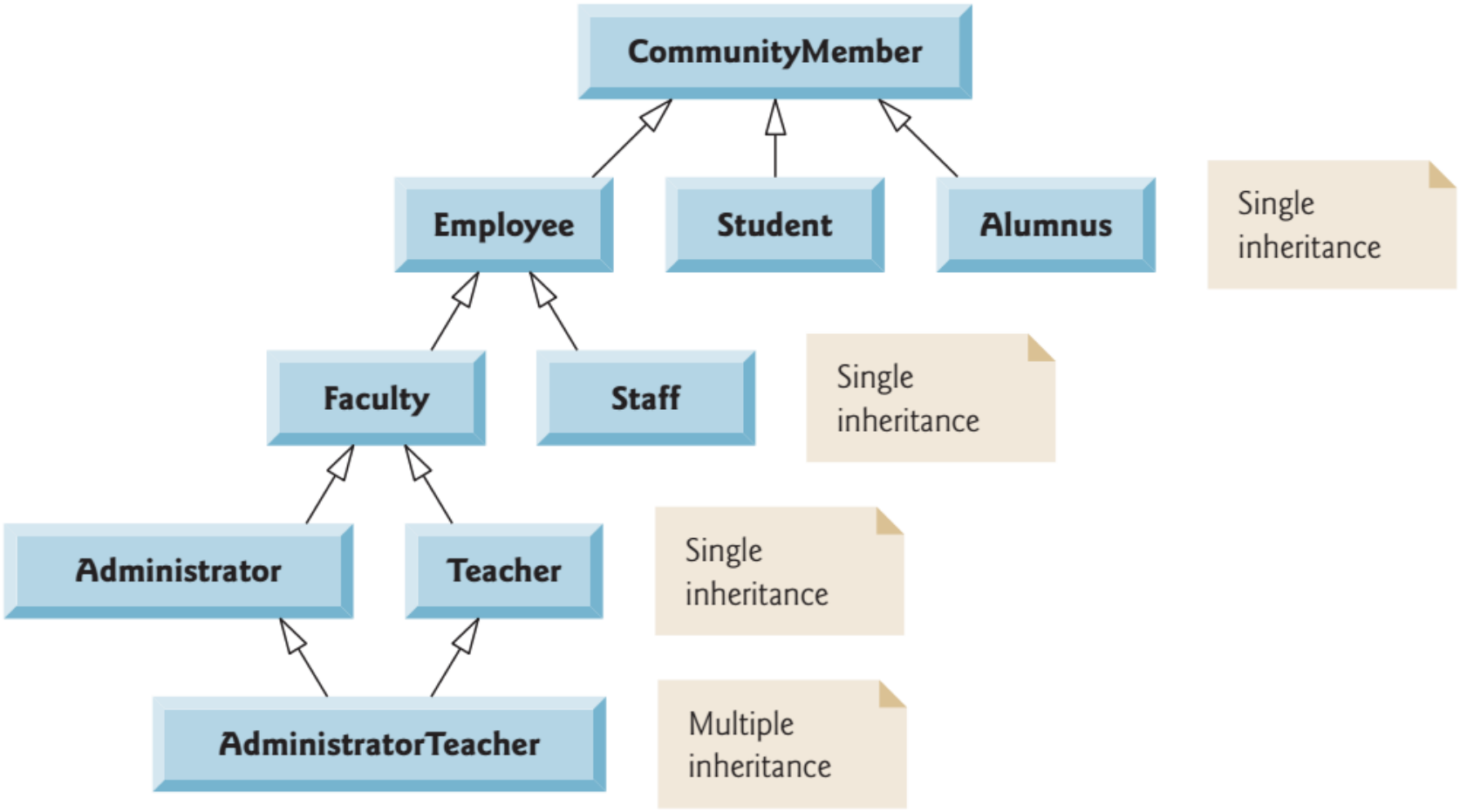
Part 2

Contents

- Constructor and Destructor in Derived Classes
- Public, Protected and Private Inheritance

Inheritance hierarchy for university CommunityMembers.

is-a relationship vs. *has-a relationship*



Inheritance

- Also, derived class can access private member of Base class through public members

```
#include <iostream>

using namespace std;

class BaseClass
{
    public:
        void setNum(int a)
        {
            num=a;
        }

        int getNum()
        {
            return num;
        }

    private:
        int num;
};

class DerivedClass: public BaseClass
{

};

int main()
{
    DerivedClass obj;
    obj.setNum(100);

    cout << "The number in BaseClass: " << obj.getNum()
       << endl;

    return 0;
}
```

Constructors and Destructors in Derived Classes

- Instantiating a derived-class object begins a chain of constructor calls in which the derived-class constructor, before performing its own tasks, invokes its direct base class's constructor.

- If the base class is derived from another class, the base-class constructor is required to invoke the constructor of the next class up in the hierarchy, and so on.

- The last constructor called in this chain is the constructor of the class at the base of the hierarchy.

- Note: Base-class *constructors, destructors and overloaded assignment operators* are not inherited by derived classes.

- Derived-class constructors, destructors and overloaded assignment operators, however, can call baseclass constructors, destructors and overloaded assignment operators.

Check the constructor and destructor order

```cpp
#include <iostream>
using namespace std;
class BaseClass
{
    public:
    BaseClass()
    {
        cout << "\nConstructor for BaseClass runs" <<endl;
    }

    ~BaseClass()
    {
        cout << "\nDestructor for BaseClass runs" <<endl;
    };
};

class DerivedClass: public BaseClass
{
    public:

    DerivedClass()
    {
        cout << "\nConstructor for DerivedClass runs" <<endl;
    }


    ~DerivedClass()
    {
        cout << "\nDestructor for DerivedClass runs" <<endl;
    };
};

int main()
{
    DerivedClass obj;
}
```

Do it Now

- Create three classes A, B, C, where class B inherits from class A and class C inherits from class B.
- Check their constructor and destructor order
- You can name class A, B, and C arbitrarily.

Practice5

- Defines class CommissionEmployee and class BasePlusCommissionEmployee with constructors and destructors that each print a message when invoked.

- Demonstrate the order in which the constructors and destructors are called for objects in an inheritance hierarchy.

Download source code for Practice5 from This Week Contents in BlackBoard

Summary of base-class member accessibility in a derived class

Base-class member-access specifier	Type of inheritance		
	`public` inheritance	`protected` inheritance	`private` inheritance
`public`	`public` in derived class. Can be accessed directly by member functions, `friend` functions and nonmember functions.	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`private` in derived class. Can be accessed directly by member functions and `friend` functions.
`protected`	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`private` in derived class. Can be accessed directly by member functions and `friend` functions.
`private`	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.

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

class B : public A
{
  // x is public
  // y is protected
  // z is not accessible from B
};
```

```
class C : protected A
{
        // x is protected
        // y is protected
        // z is not accessible from C
};


class D : private A
// 'private' is default for classes
{
        // x is private
        // y is private
        // z is not accessible from D
};
```

Example: Inheritance in a glance

Next Class

- OOP: Polymorphism in C++

Notice

- Discussion on Homework2
- Discussion on the supplementary Lecture