CSE 230 Intermediate Programming in C and C++

Inheritance

Fall 2017

Stony Brook University

Instructor: Shebuti Rayana

http://www3.cs.stonybrook.edu/~cse230/

Ref. Book: C How to Program, 8th edition by Deitel and Deitel

Introduction to Inheritance

- A form of software reusability in which new classes are created from the existing classes by absorbing their attributes and behaviors and overriding or embellishing these with capabilities the new class requires.
- When creating a new class, instead of writing new members, the programmer can designate that the new class is to *inherit* a previously defined ***base class***.
- The new class is referred to as ***derived class***.
- Each derived class itself becomes candidate to be a base class for some future derived class.
- With *single inheritance*, a class is derived from one base class. With *multiple inheritance*, a class is derived from multiple (possibly unrelated) base classes.

Base class and Derived class

- Often an object of one class “is an” object of another class as well.
- A rectangle certainly is a quadrilateral (as is square, trapezoid and a parallelogram). Thus the class **Rectangle** can be said to inherit from class **Quadrilateral**.
- In this context, class **Quadrilateral** is called the *base class* and class **Rectangle** is called a *derived class*.
- Java uses different terminology: The base class is called the *superclass* (superset of objects) and the derived class is called the *subclass* (subset of objects).
- Inheritance normally produces derived classes with *more* features than their base classes, so the terms superclass and subclass may be confusing.

Inheritance Examples

Base class	Derived classes
Student	GraduateStudent
	UndergraduateStudent
Shape	Circle
	Triangle
	Rectangle
Loan	CarLoan
	HomeImprovementLoan
	MortgageLoan
Employee	FacultyMember
	StaffMember
Account	CheckingAccount
	SavingsAccount

Types of Inheritance

- A derived class can add data members and member functions of its own, so a derived class can be larger than its base class.
- A derived class is more specific than its base class.
- The derived class has the ability to add, replace or refine the features of the base class.
- C++ offers three kinds of inheritance – **public, protected** and **private**.
- With **public** inheritance, every object of a derived class may also be treated as an object of that derived class's base class. However, the converse is not true – base-class objects are not objects of that base class's derived classes.

Protected Members

- A base class's **public** members are accessible by all functions in the program.
- A base class's **private** members are accessible only by member functions and **friend**s of the base class.
- The **protected** access is an intermediate level of protection between **public** access and **private** access.
- A base class's **protected** members may be accessed only by members and **friend**s of the base class and by members and **friend**s of derived classes.
- Derived-class members can refer to **public** and **protected** members of the base class simply by using member names.
- A **protected** data "breaks" encapsulation.

Inheritance: Example

```
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
};
```

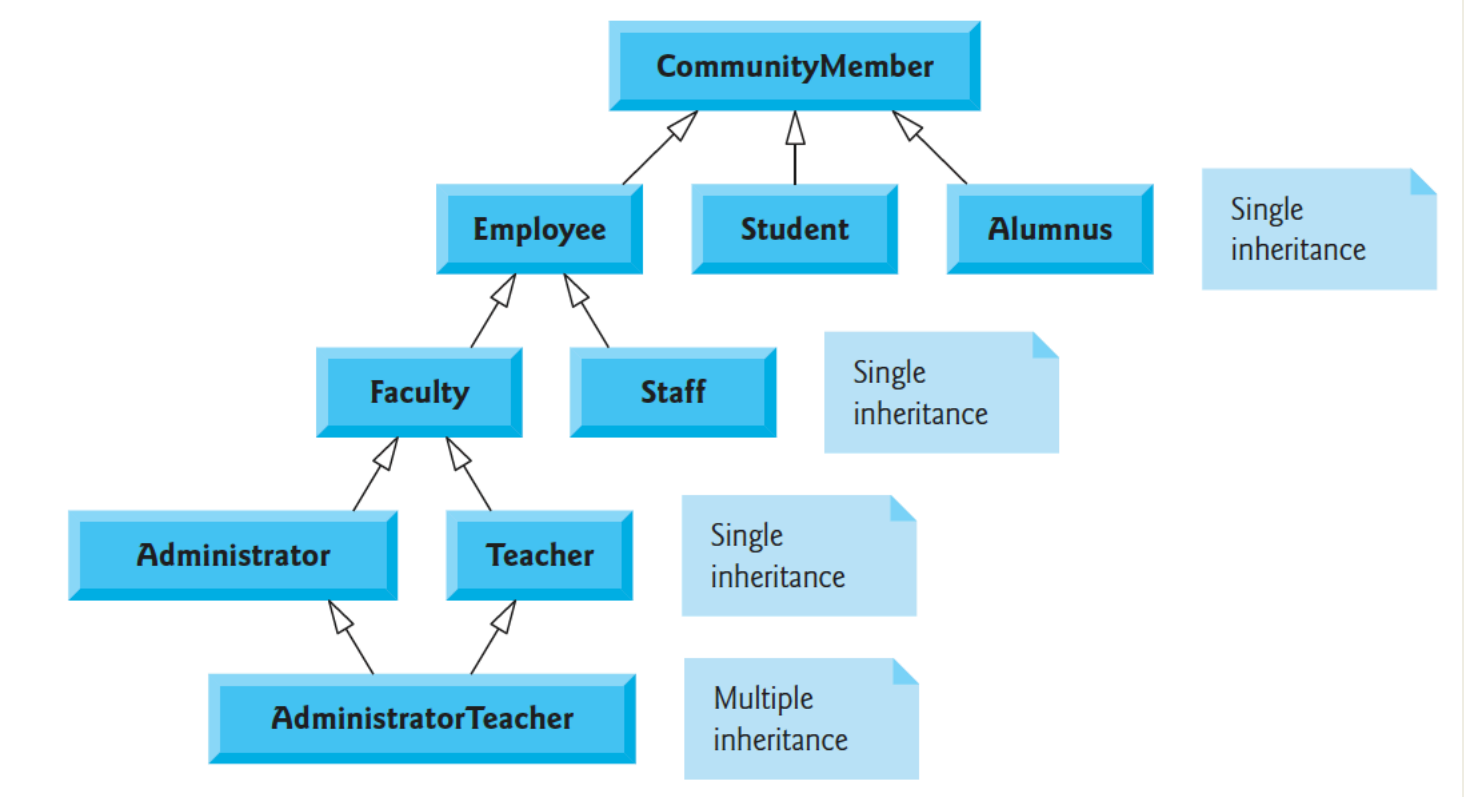
What is inherited from the base class?

- In principle, a publicly derived class inherits access to every member of a base class except:
  - its constructors and its destructor
  - its assignment operator members (operator=)
  - its friends
  - its private members
- Even though access to the constructors and destructor of the base class is not inherited as such, they are automatically called by the constructors and destructor of the derived class.

Inheritance and Hierarchical Structures

- Inheritance forms tree-like hierarchical structures.
- A base class exists in a hierarchical structure with its derived classes.
- A class can certainly exist by itself, but it is when a class is used with the mechanism of inheritance that the class becomes either a base class that supplies attributes and behaviors to other classes, or the class becomes a derived class that inherits attributes and behaviors.

Inheritance Hierarchy for University Members



Overriding Base-Class Members

- A derived class can override a base-class member function by supplying a new version of that function with the same signature.
- If the signatures are different, this would be function overloading rather than function overriding.
- When that function is mentioned by name in the derived class, the derived-class version is selected.
- The scope resolution operator may be used to access the base-class version from the derived class.

Class Object Conversion

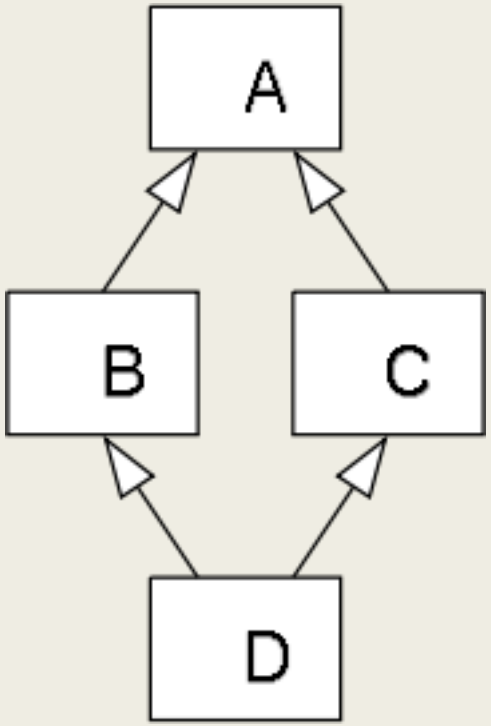
- Despite the fact that a derived-class object also *is a* base-class object, the derived-class type objects and the base-class type objects are different.
- Under public inheritance, derived-class objects can be treated as base-class objects. Remember, derived class objects can have more members than the base class.
- After a derived-class object is assigned to a base-class object, referring to derived-class-only members is an error.
- Base-class objects can not be assigned to derived-class objects (unless assignment operator is properly overloaded) because it would leave the additional derived-class members undefined.

Composition vs. Inheritance

- We distinguish between *is-a relationships* and *has-a relationships*.
- *is-a relationships* is inheritance. In an *is a* relationship, an object of a derived-class type may also be treated as on abject of the base-class type.
- *has-a relationships* is composition. In a *has a* relationship, a class object *has* one or more objects of other classes as members.
- For example, given the classes **Employee**, **BirthDate** and **TelephoneNumber**, it is improper to say that an Employee *is a* **TelephoneNumber**. But it is certainly appropriate to say that each **Employee** *has a* **BirthDate** and each **Employee** *has a* **TelephoneNumber**.

The Diamond Problem

- The "diamond problem" (sometimes referred to as the "**deadly diamond of death**") is an ambiguity that arises when two classes B and C inherit from A, and class D inherits from both B and C.
- There will be two copies of A's members in D, one through B and one through C.



To solve this virtual Inheritance is used

Virtual Inheritance

- Fortunately, C++ allows us to solve this problem by using virtual inheritance.
- In order to prevent the compiler from giving an error we use the keyword **virtual** when we inherit from the base class in both derived classes.