Introduction to Inheritance

Chapter 9

Objectives

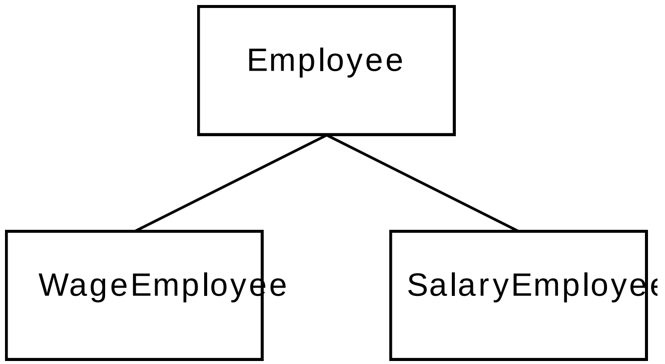
- **Use inheritance to model your problem domain and achieve greater code reuse.**
- **Use C++ class derivation to implement inheritance.**
- **Use *public*, *protected* and *private* to control access to class members.**
- **Use an initialization list for proper base class initialization and embedded member initialization.**
- **Determine order of invocation of constructors and destructors.**
- **Distinguish between use of inheritance and composition.**
- **Gain experience through code walk-throughs and lab exercises.**

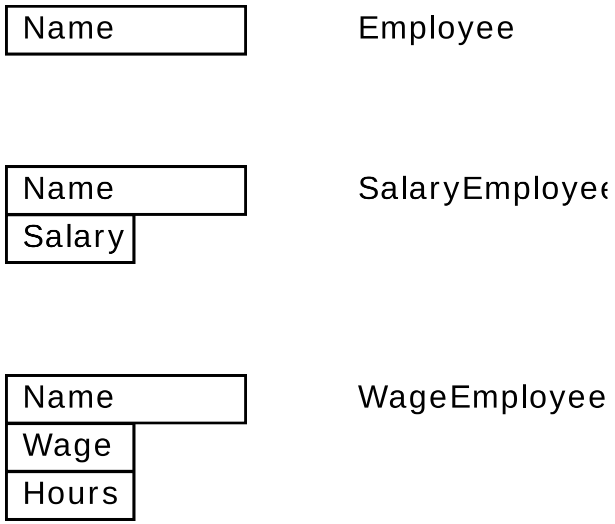
  - The example programs are in the **chapter directory.**
  - Labs located in Labs/Lab9

Inheritance Concept

- **Inheritance is a key feature of the object-oriented programming paradigm.**
  - You abstract out common features of your classes and put them in a high-level base class.
  - You can add or change features in more specialized derived classes, which "inherit" the standard behavior from the base class.
  - Inheritance facilitates code reuse and extensibility.
- **Consider *Employee* as a base class, with derived classes *WageEmployee* and *SalaryEmployee*.**
  - All employees share some attributes, such as name.
  - Wage employees and salaried employees differ in other respects, such as in how their pay is computed.

Inheritance Hierarchy





Inheritance in C++

- **Inheritance is implemented in C++ by a mechanism known as *class derivation*:**

```
class  DerivedClass : public  BaseClass
{ ... };
```

- **Base class must be declared prior to the derived class.**
- **DerivedClass can use all public (and protected) members of BaseClass, but it does not have any special access to the private members of BaseClass.**
  - **If a derived class did have access to private members of its base class, the access security could be defeated simply by deriving a class!**

Employee demo

- **The folder Employee contains a starting point to examine inheritance.**
- **There is one base class Employee, and two derived classes, SalaryEmployee and WageEmployee.**
- **Examine the code in the header file Employee.h and the implementation file DemoEmp.cpp.**
- **Build and run the program.**

Protected Members

- **So far we have seen two access privileges: public and private.**
- **Class derivation introduces a different kind of user: the derived class.**
    - **SalaryEmployee** is derived from **Employee** but has no special privileges to access the private members of **Employee.**
- **To allow special privilege for this user, protected access privilege is provided as the third type of access privilege.**
    - Since **m_name** is declared as **protected** in the **Employee** base class, the derived class could access it, but classes not derived from **Employee** could not.
- **Members specified as protected become public to the derived class, but remain private to all other classes and program.**
- **Rules for private and public are same for the derived classes.**

Base Class Initializer List

- **When the base class constructor requires arguments, the arguments are passed via an "initialization list"**

```
class SalaryEmployee : public Employee{
public:
   SalaryEmployee(const char *name, int salary)
           ...
};
```

Base Class Initializer List (continued)

- **Here an initializer list is used in the constructor to pass arguments to the base class constructor for *Employee* (name is passed to Employee c’tor):**

```
class SalaryEmployee : public Employee {
public:
   SalaryEmployee(const char *name, int salary) : Employee(name) {
         m_salary = salary;
      }
          ...
};
```

Composition

- **Another way for a new class to reuse code is to simply create an object of the other class inside the new class.**
    - This technique is called **composition**.
- ***Employee* could use a *String* object to represent employee name.**

```
class Employee{
public:
    Employee(const char *name = ""){m_name = name;}
    void SetName(const char *name) { m_name = name;}
    const char* GetName() const {return m_name;}
private:
    String m_name;
};
```

Base class default constructor

- **If you don't do anything special, the compiler will generate code to implicitly call the default constructor for the member object before constructing the containing object.**
- **If a default constructor does not exist and you do not explicitly call one of the non-default constructors in the base you will receive an error at compile time.**

Member Initializer List

- **A better approach is to use a "member initializer list” "**, which has similar syntax to a base **class initializer list:**

  Employee::Employee(const char* name) : **m_name(name)** { }

- **This syntax causes the *String* class constructor to be invoked with the argument *name*.**
- **The *String* class constructor is called first before the *Employee* constructor starts executing.**
- **The member object get data assigned exactly once.**
- **The same syntax can also be used for built-in data types, and member object initialization and base class initialization can be combined.**

Order of Initialization

- **C++ has a defined order for the construction and destruction of base class objects, derived class objects, and member objects.**
- **It is important to know this order in cases where there are interdependencies among classes.**
  - You should avoid a situation where an object gets prematurely destroyed while another object refers to its data.

Order of Initialization (continued)

- **The order of construction is:**
  - Constructor of *BaseClass*
  - Constructor of *member1*
  - Constructor of *member2*
  - Constructor of *DerivedClass*
- **Destructors are invoked in exact reverse order.**

```
class DerivedClass : public BaseClass{
public:
    member1;
    member2;
};
```

Inheritance vs. Composition

- **Inheritance and composition are both code reuse techniques in which data from one class is contained within another class.**
  - When do you prefer one technique over the other?
- **Inheritance is used when an “Is-A” relationship exists:**
  - A SalaryEmployee is an Employee
  - The derived class supports the same interface as the base class, plus some additional features
- **Composition is used when a “Has-A” relationship exists:**
  - **Employee** has a **String** as a data member to represent the name
  - Composition is suitable when you when you want the features of another class but not its interface

Summary

- C+ + has special features to allow class inheritance, which allows you to better model your problem domain and to achieve greater code reuse.
- Members of a base class are also members of derived classes.
- Protected members of a base class can be accessed by derived classes but not by any other classes.
- Initialization lists can be used to properly initialize member objects and base class objects.
- The order of invoking constructors is from the base class to the derived class.
- Inheritance models “Is-A” relationships and composition models “Has-A” relationships.