Software Engineering Observation 11.1

Copying and pasting code from one class to another can spread many physical copies of the same code and can spread errors throughout a system, creating a code-maintenance nightmare. To avoid duplicating code (and possibly errors), use inheritance, rather than the "copy-and-paste" approach, in situations where you want one class to "absorb" the data members and member functions of another class.

Software Engineering Observation 11.2

With inheritance, the common data members and member functions of all the classes in the hierarchy are declared in a base class. When changes are required for these common features, you need to make the changes only in the base class—derived classes then inherit the changes. Without inheritance, changes would need to be made to all the source code files that contain a copy of the code in question.

11.3.3 Creating a `CommissionEmployee`–`BasePlusCommissionEmployee` Inheritance Hierarchy

- Now we create and test a new `BasePlusCommissionEmployee` class (Figs. 11.10–11.11) that derives from class `CommissionEmployee` (Figs. 11.4–11.5).
- In this example, a `BasePlusCommissionEmployee` object *is a* `CommissionEmployee` (because inheritance passes on the capabilities of class `CommissionEmployee`), but class `BasePlusCommissionEmployee` also has data member `baseSalary` (Fig. 11.10, line 22).
- The colon (`:`) in line 10 of the class definition indicates inheritance.
- Keyword `public` indicates the *type of inheritance*.
- As a derived class (formed with `public` inheritance), `BasePlusCommissionEmployee` inherits all the members of class `CommissionEmployee`, except for the constructor—each class provides its own constructors that are specific to the class.

11.3.3 Creating a `CommissionEmployee`–`BasePlusCommissionEmployee` Inheritance Hierarchy (cont.)

- Destructors, too, are not inherited
- Thus, the `public` services of `BasePlusCommissionEmployee` include its constructor and the `public` member functions inherited from class *`CommissionEmployee`—although we cannot see these inherited member functions* in `BasePlusCommissionEmployee`'s source code, they're nevertheless a part of derived class `BasePlusCommissionEmployee`.
- The derived class's `public` services also include member functions `setBaseSalary`, `getBaseSalary`, `earnings` and `print`.

```cpp
// Fig. 11.10: BasePlusCommissionEmployee.h
// BasePlusCommissionEmployee class derived from class
// CommissionEmployee.
#ifndef BASEPLUS_H
#define BASEPLUS_H

#include <string> // C++ standard string class
#include "CommissionEmployee.h" // CommissionEmployee class declaration

class BasePlusCommissionEmployee : public CommissionEmployee
{
public:
   BasePlusCommissionEmployee( const std::string &, const std::string &,
      const std::string &, double = 0.0, double = 0.0, double = 0.0 );

   void setBaseSalary( double ); // set base salary
   double getBaseSalary() const; // return base salary

```

Fig. 11.10 | `BasePlusCommissionEmployee` class definition indicating inheritance relationship with class `CommissionEmployee`. (Part 1 of 2.)

```
    double earnings() const; // calculate earnings
    void print() const; // print BasePlusCommissionEmployee object
private:
    double baseSalary; // base salary
}; // end class BasePlusCommissionEmployee

#endif
```

Fig. 11.10 | BasePlusCommissionEmployee class definition indicating inheritance relationship with class CommissionEmployee. (Part 2 of 2.)

```cpp
// Fig. 11.11: BasePlusCommissionEmployee.cpp
// Class BasePlusCommissionEmployee member-function definitions.
#include <iostream>
#include <stdexcept>
#include "BasePlusCommissionEmployee.h"
using namespace std;

// constructor
BasePlusCommissionEmployee::BasePlusCommissionEmployee(
   const string &first, const string &last, const string &ssn,
   double sales, double rate, double salary )
   // explicitly call base-class constructor
   : CommissionEmployee( first, last, ssn, sales, rate )
{
   setBaseSalary( salary ); // validate and store base salary
} // end BasePlusCommissionEmployee constructor

```

Fig. 11.11 | `BasePlusCommissionEmployee` implementation file: `private` base-class data cannot be accessed from derived class. (Part 1 of 4.)

```
// set base salary
void BasePlusCommissionEmployee::setBaseSalary( double salary )
{
   if ( salary >= 0.0 )
      baseSalary = salary;
   else
      throw invalid_argument( "Salary must be >= 0.0" );
} // end function setBaseSalary

// return base salary
double BasePlusCommissionEmployee::getBaseSalary() const
{
   return baseSalary;
} // end function getBaseSalary

// calculate earnings
double BasePlusCommissionEmployee::earnings() const
{
   // derived class cannot access the base class's private data
   return baseSalary + ( commissionRate * grossSales );
} // end function earnings

```

Fig. 11.11 | BasePlusCommissionEmployee implementation file: private base-class data cannot be accessed from derived class. (Part 2 of 4.)

```cpp
// print BasePlusCommissionEmployee object
void BasePlusCommissionEmployee::print() const
{
   // derived class cannot access the base class's private data
   cout << "base-salaried commission employee: " << firstName << ' '
      << lastName << "\nsocial security number: " << socialSecurityNumber
      << "\ngross sales: " << grossSales
      << "\ncommission rate: " << commissionRate
      << "\nbase salary: " << baseSalary;
} // end function print
```

Fig. 11.11 | `BasePlusCommissionEmployee` implementation file: `private` base-class data cannot be accessed from derived class. (Part 3 of 4.)

Compilation Errors from the LLVM Compiler in Xcode 4.5

```
BasePlusCommissionEmployee.cpp:37:26:
   'commissionRate' is a private member of 'CommissionEmployee'
BasePlusCommissionEmployee.cpp:37:43:
   'grossSales' is a private member of 'CommissionEmployee'
BasePlusCommissionEmployee.cpp:44:53:
   'firstName' is a private member of 'CommissionEmployee'
BasePlusCommissionEmployee.cpp:45:10:
   'lastName' is a private member of 'CommissionEmployee'
BasePlusCommissionEmployee.cpp:45:54:
   'socialSecurityNumber' is a private member of 'CommissionEmployee'
BasePlusCommissionEmployee.cpp:46:31:
   'grossSales' is a private member of 'CommissionEmployee'
BasePlusCommissionEmployee.cpp:47:35:
   'commissionRate' is a private member of 'CommissionEmployee'
```

Fig. 11.11 | `BasePlusCommissionEmployee` implementation file: `private` base-class data cannot be accessed from derived class. (Part 4 of 4.)

11.3.3 Creating a `CommissionEmployee`–`BasePlusCommissionEmployee` Inheritance Hierarchy (cont.)

- Figure 11.11 shows `BasePlusCommissionEmployee`'s member-function implementations.
- The constructor introduces base-class initializer syntax, which uses a member initializer to pass arguments to the base-class constructor.
- C++ requires that a derived-class constructor call its base-class constructor to initialize the base-class data members that are inherited into the derived class.
- If `BasePlusCommissionEmployee`'s constructor did not invoke class `CommissionEmployee`'s constructor *explicitly*, C++ would attempt to invoke class `CommissionEmployee`'s default constructor—but the class does not have such a constructor, so the compiler would issue an error.

Common Programming Error 11.1

When a derived-class constructor calls a base-class constructor, the arguments passed to the base-class constructor must be consistent with the number and types of parameters specified in one of the base-class constructors; otherwise, a compilation error occurs.

Performance Tip 11.1

In a derived-class constructor, invoking base-class constructors and initializing member objects explicitly in the member initializer list prevents duplicate initialization in which a default constructor is called, then data members are modified again in the derived-class constructor's body.

11.3.3 Creating a `CommissionEmployee`–`BasePlusCommissionEmployee` Inheritance Hierarchy (cont.)

Compilation Errors from Accessing Base-Class `private` Members

- The compiler generates errors for line 37 of Fig. 11.11 because base class `CommissionEmployee`'s data members `commissionRate` and `grossSales` are `private`—derived class `BasePlusCommissionEmployee`'s member functions are *not* allowed to access base class `CommissionEmployee`'s `private` data.
- We used red text in Fig. 11.11 to indicate erroneous code.
- The compiler issues additional errors in lines 44–47 of `BasePlus-Commission-Employee`'s `print` member function for the same reason.
- C++ rigidly enforces restrictions on accessing `private` data members, so that *even a derived class (which is intimately related to its base class) cannot access the base class's `private` data.*

11.3.3 Creating a `CommissionEmployee`–`BasePlusCommissionEmployee` Inheritance Hierarchy (cont.)

Preventing the Errors in ***`BasePlusCommissionEmployee`***

- We purposely included the erroneous code in Fig. 11.11 to emphasize that a derived class's member functions cannot access its base class's `private` data.
- The errors in `BasePlusCommissionEmployee` could have been prevented by using the *get* member functions inherited from class `CommissionEmployee`.