11.3.5 CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy Using private Data (cont.)

BasePlusCommissionEmployee Member Function print

- BasePlusCommissionEmployee's print function (Fig. 11.15, lines 40–48) redefines class CommissionEmployee's print function (Fig. 11.14, lines 91–98) to output the appropriate base-salaried commission employee information.
- By using inheritance and by calling member functions that hide the data and ensure consistency, we've efficiently and effectively constructed a well-engineered class.

11.4 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 either explicitly (via a base-class member initializer) or implicitly (calling the base class's default 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, whose body actually finishes executing *first*.
- The most derived-class constructor's body finishes executing *last*.
- Each base-class constructor initializes the base-class data members that the derived-class object inherits.

Software Engineering Observation 11.6

When a program creates a derived-class object, the derived-class constructor immediately calls the base-class constructor, the base-class constructor's body executes, then the derived class's member initializers execute and finally the derived-class constructor's body executes. This process cascades up the hierarchy if it contains more than two levels.

11.4 Constructors and Destructors in Derived Classes (cont.)

- When a derived-class object is destroyed, the program calls that object's destructor.
- This begins a chain (or cascade) of destructor calls in which the derived-class destructor and the destructors of the direct and indirect base classes and the classes' members execute in *reverse* of the order in which the constructors executed.
- When a derived-class object's destructor is called, the destructor performs its task, then invokes the destructor of the next base class up the hierarchy.
- This process repeats until the destructor of the final base class at the top of the hierarchy is called.
- Then the object is removed from memory.

Software Engineering Observation 11.7

Suppose that we create an object of a derived class where both the base class and the derived class contain (via composition) objects of other classes. When an object of that derived class is created, first the constructors for the base class's member objects execute, then the base-class constructor body executes, then the constructors for the derived class's member objects execute, then the derived class's constructor body executes. Destructors for derived-class objects are called in the reverse of the order in which their corresponding constructors are called.

11.4 Constructors and Destructors in Derived Classes (cont.)

- Base-class constructors, destructors and overloaded assignment operators (Chapter 10) are *not* inherited by derived classes.
- Derived-class constructors, destructors and overloaded assignment operators, however, can call base-class versions.

11.4 Constructors and Destructors in Derived Classes (cont.)

C++11: Inheriting Base Class Constructors

- Sometimes a derived class's constructors simply mimic the base class's constructors.
- A frequently requested convenience feature for C++11 was the ability to *inherit* a base class's constructors.
- You can now do this by explicitly including a using declaration of the form

```
using BaseClass::BaseClass;
```

- *anywhere* in the derived-class definition.
- In the preceding declaration, `BaseClass` is the base class's name.

11.4 Constructors and Destructors in Derived Classes (cont.)

- When you inherit constructors:
  - By default, each inherited constructor has the *same* access level (`public`, `protected` or `private`) as its corresponding base-class constructor.
  - The default, copy and move constructors are *not* inherited.
  - If a constructor is *deleted* in the base class by placing `= delete` in its prototype, the corresponding constructor in the derived class is *also* deleted.
  - If the derived class does not explicitly define constructors, the compiler generates a default constructor in the derived class—*even* if it inherits other constructors from its base class.
  - If a constructor that you *explicitly* define in a derived class has the *same* parameter list as a base-class constructor, then the base-class constructor is *not* inherited.
  - A base-class constructor's default arguments are *not* inherited. Instead, the compiler generates overloaded constructors in the derived class.

11.5 `public`, `protected` and `private` Inheritance

- When deriving a class from a base class, the base class may be inherited through `public`, `protected` or `private` inheritance.
- Use of `protected` and `private` inheritance is rare.
- Figure 11.16 summarizes for each type of inheritance the accessibility of base-class members in a derived class.
- The first column contains the base-class access specifiers.
- 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.

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.

11.6 Software Engineering with Inheritance

- When we use inheritance to create a new class from an existing one, the new class inherits the data members and member functions of the existing class, as described in Fig. 11.16.
- We can customize the new class to meet our needs by including additional members and by redefining base-class members.
- The derived-class programmer does this in C++ without accessing the base class's source code.
- The derived class must be able to link to the

11.6 Software Engineering with Inheritance (cont.)

- When we use inheritance to create a new class from an existing one, the new class inherits the data members and member functions of the existing class.
- We can customize the new class to meet our needs by redefining base-class members and by including additional members.
- The derived-class programmer does this in C++ without accessing the base class's source code (the derived class must be able to link to the base class's object code).

11.7 Software Engineering with Inheritance (cont.)

- Software developers can develop proprietary classes for sale or license.
- Users then can derive new classes from these library classes rapidly and without accessing the proprietary source code.
- The software developers need to supply the headers along with the object code
- The availability of substantial and useful class libraries delivers the maximum benefits of software reuse through inheritance.

Software Engineering Observation 11.8

At the design stage in an object-oriented system, the designer often determines that certain classes are closely related. The designer should "factor out" common attributes and behaviors and place these in a base class, then use inheritance to form derived classes.

Software Engineering Observation 11.9

Creating a derived class does not affect its base class's source code. Inheritance preserves the integrity of the base class.