Good Programming Practice 12.1

Even though certain functions are implicitly `virtual` because of a declaration made higher in the class hierarchy, explicitly declare these functions `virtual` at every level of the class hierarchy to promote program clarity.

Software Engineering Observation 12.6

When a derived class chooses not to override a `virtual` function from its base class, the derived class simply inherits its base class's `virtual` function implementation.

12.3.4 Virtual Functions and Virtual Destructors (cont.)

Invoking a `virtual` Function Through a Base-Class Pointer or Reference

- If a program invokes a `virtual` function through a base-class pointer to a derived-class object (e.g., `shapePtr->draw()`) or a base-class reference to a derived-class object (e.g., `shapeRef.draw()`), the program will choose the correct derived-class function dynamically (i.e., at execution time) *based on the object type—not the pointer or reference type*.
  - Known as dynamic binding or late binding.

12.3.4 Virtual Functions and Virtual Destructors (cont.)

Invoking a `virtual` Function Through an Object's Name

- When a `virtual` function is called by referencing a specific object by name and using the dot member-selection operator (e.g., `squareObject.draw()`), the function invocation is re-solved at compile time (this is called static binding) and the `virtual` function that is called is the one defined for (or inherited by) the class of that particular object—this is *not* polymorphic behavior.
- Dynamic binding with `virtual` functions occurs only off pointers (and, as we'll soon

12.3.4 Virtual Functions and Virtual Destructors (cont.)

***virtual* Functions in the `CommissionEmployee` Hierarchy**

- Figures 12.4–12.5 are the headers for classes `CommissionEmployee` and `BasePlusCommissionEmployee`, respectively.
- We modified these to declare each class's `earnings` and `print` member functions as `virtual` (lines 29–30 of Fig. 12.4 and lines 19–20 of Fig. 12.5).
- Because functions `earnings` and `print` are `virtual` in class `CommissionEmployee`, class `BasePlusCommissionEmployee`'s `earnings` and `print` functions *override* class `CommissionEmployee`'s.
- In addition, class `BasePlusCommissionEmployee`'s `earnings` and `print` functions are declare `override`.

Error-Prevention Tip 12.1

Apply C++11's `override` keyword to every overridden function in a derived-class. This forces the compiler to check whether the base class has a member function with the same name and parameter list (i.e., the same signature). If not, the compiler generates an error.

```
1  // Fig. 12.4: CommissionEmployee.h
2  // CommissionEmployee class header declares earnings and print as virtual.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // C++ standard string class
7
8  class CommissionEmployee
9  {
10 public:
11    CommissionEmployee( const std::string &, const std::string &,
12       const std::string &, double = 0.0, double = 0.0 );
13
14    void setFirstName( const std::string & ); // set first name
15    std::string getFirstName() const; // return first name
16
17    void setLastName( const std::string & ); // set last name
18    std::string getLastName() const; // return last name
19
20    void setSocialSecurityNumber( const std::string & ); // set SSN
21    std::string getSocialSecurityNumber() const; // return SSN
22
```

Fig. 12.4 | CommissionEmployee class header declares earnings and print as virtual.

```
23      void setGrossSales( double ); // set gross sales amount
24      double getGrossSales() const; // return gross sales amount
25
26      void setCommissionRate( double ); // set commission rate
27      double getCommissionRate() const; // return commission rate
28
29      virtual double earnings() const; // calculate earnings
30      virtual void print() const; // print object
31   private:
32      std::string firstName;
33      std::string lastName;
34      std::string socialSecurityNumber;
35      double grossSales; // gross weekly sales
36      double commissionRate; // commission percentage
37   }; // end class CommissionEmployee
38
39   #endif
```

Fig. 12.4 | CommissionEmployee class header declares `earnings` and `print` as `virtual`.

```
1  // Fig. 12.5: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class derived from class
3  // CommissionEmployee.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // C++ standard string class
8  #include "CommissionEmployee.h" // CommissionEmployee class declaration
9
10 class BasePlusCommissionEmployee : public CommissionEmployee
11 {
12 public:
13    BasePlusCommissionEmployee( const std::string &, const std::string &,
14       const std::string &, double = 0.0, double = 0.0, double = 0.0 );
15
16    void setBaseSalary( double ); // set base salary
17    double getBaseSalary() const; // return base salary
18
```

Fig. 12.5 | BasePlusCommissionEmployee class header declares `earnings` and `print` functions as `virtual` and `override`. (Part 1 of 2.)

```
19      virtual double earnings() const override; // calculate earnings
20      virtual void print() const override; // print object
21   private:
22      double baseSalary; // base salary
23   }; // end class BasePlusCommissionEmployee
24
25   #endif
```

Fig. 12.5 | `BasePlusCommissionEmployee` class header declares `earnings` and `print` functions as `virtual` and `override`. (Part 2 of 2.)

12.3.4 Virtual Functions and Virtual Destructors (cont.)

- We modified Fig. 12.1 to create the program of Fig. 12.6.
- Lines 40–51 of Fig. 12.6 demonstrate again that a `CommissionEmployee` pointer aimed at a `CommissionEmployee` object can be used to invoke `CommissionEmployee` functionality, and a `BasePlusCommissionEmployee` pointer aimed at a `BasePlusCommissionEmployee` object can be used to invoke `BasePlusCommissionEmployee` functionality.
- Line 54 aims base-class pointer `commissionEmployeePtr` at derived-class object `basePlusCommissionEmployee`.
- Note that when line 61 invokes member function `print` off the base-class pointer, the derived-class `BasePlusCommissionEmployee`'s `print` member function is invoked, so line 61 outputs different text than line 53 does in Fig. 12.1 (when member function `print` was not declared `virtual`).
- We see that declaring a member function `virtual` causes the program to dynamically determine which function to invoke *based on the type of object to which the handle points, rather than on the type of the handle*.

```
1  // Fig. 12.6: fig12_06.cpp
2  // Introducing polymorphism, virtual functions and dynamic binding.
3  #include <iostream>
4  #include <iomanip>
5  #include "CommissionEmployee.h"
6  #include "BasePlusCommissionEmployee.h"
7  using namespace std;
8
9  int main()
10 {
11    // create base-class object
12    CommissionEmployee commissionEmployee(
13       "Sue", "Jones", "222-22-2222", 10000, .06 );
14
15    // create base-class pointer
16    CommissionEmployee *commissionEmployeePtr = nullptr;
17
18    // create derived-class object
19    BasePlusCommissionEmployee basePlusCommissionEmployee(
20       "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
21
```

Fig. 12.6 | Demonstrating polymorphism by invoking a derived-class `virtual` function via a base-class pointer to a derived-class object. (Part 1 of 5.)

```
22      // create derived-class pointer
23      BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = nullptr;
24
25      // set floating-point output formatting
26      cout << fixed << setprecision( 2 );
27
28      // output objects using static binding
29      cout << "Invoking print function on base-class and derived-class "
30         << "\nobjects with static binding\n\n";
31      commissionEmployee.print(); // static binding
32      cout << "\n\n";
33      basePlusCommissionEmployee.print(); // static binding
34
35      // output objects using dynamic binding
36      cout << "\n\n\nInvoking print function on base-class and "
37         << "derived-class \nobjects with dynamic binding";
38
39      // aim base-class pointer at base-class object and print
40      commissionEmployeePtr = &commissionEmployee;
41      cout << "\n\nCalling virtual function print with base-class pointer"
42         << "\nto base-class object invokes base-class "
43         << "print function:\n\n";
44      commissionEmployeePtr->print(); // invokes base-class print
```

Fig. 12.6 | Demonstrating polymorphism by invoking a derived-class `virtual` function via a base-class pointer to a derived-class object. (Part 2 of 5.)

```
45
46      // aim derived-class pointer at derived-class object and print
47      basePlusCommissionEmployeePtr = &basePlusCommissionEmployee;
48      cout << "\n\nCalling virtual function print with derived-class "
49         << "pointer\nto derived-class object invokes derived-class "
50         << "print function:\n\n";
51      basePlusCommissionEmployeePtr->print(); // invokes derived-class print
52
53      // aim base-class pointer at derived-class object and print
54      commissionEmployeePtr = &basePlusCommissionEmployee;
55      cout << "\n\nCalling virtual function print with base-class pointer"
56         << "\nto derived-class object invokes derived-class "
57         << "print function:\n\n";
58
59      // polymorphism; invokes BasePlusCommissionEmployee's print;
60      // base-class pointer to derived-class object
61      commissionEmployeePtr->print();
62      cout << endl;
63   } // end main
```

Fig. 12.6 | Demonstrating polymorphism by invoking a derived-class `virtual` function via a base-class pointer to a derived-class object. (Part 3 of 5.)

```
Invoking print function on base-class and derived-class
objects with static binding

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00


Invoking print function on base-class and derived-class
objects with dynamic binding

Calling virtual function print with base-class pointer
to base-class object invokes base-class print function:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06
```

Fig. 12.6 | Demonstrating polymorphism by invoking a derived-class `virtual` function via a base-class pointer to a derived-class object. (Part 4 of 5.)