```
22      // keyword virtual signals intent to override
23      virtual double earnings() const override; // calculate earnings
24      virtual void print() const override; // print object
25   private:
26      double grossSales; // gross weekly sales
27      double commissionRate; // commission percentage
28   }; // end class CommissionEmployee
29
30   #endif // COMMISSION_H
```

Fig. 12.13 | `CommissionEmployee` class header. (Part 2 of 2.)

```
1  // Fig. 12.14: CommissionEmployee.cpp
2  // CommissionEmployee class member-function definitions.
3  #include <iostream>
4  #include <stdexcept>
5  #include "CommissionEmployee.h" // CommissionEmployee class definition
6  using namespace std;
7
8  // constructor
9  CommissionEmployee::CommissionEmployee( const string &first,
10    const string &last, const string &ssn, double sales, double rate )
11    : Employee( first, last, ssn )
12 {
13    setGrossSales( sales );
14    setCommissionRate( rate );
15 } // end CommissionEmployee constructor
16
17 // set gross sales amount
18 void CommissionEmployee::setGrossSales( double sales )
19 {
20    if ( sales >= 0.0 )
21       grossSales = sales;
22    else
23       throw invalid_argument( "Gross sales must be >= 0.0" );
24 } // end function setGrossSales
```

Fig. 12.14 | CommissionEmployee class implementation file. (Part 1 of 3.)

```
25
26  // return gross sales amount
27  double CommissionEmployee::getGrossSales() const
28  {
29     return grossSales;
30  } // end function getGrossSales
31
32  // set commission rate
33  void CommissionEmployee::setCommissionRate( double rate )
34  {
35     if ( rate > 0.0 && rate < 1.0 )
36        commissionRate = rate;
37     else
38        throw invalid_argument( "Commission rate must be > 0.0 and < 1.0" );
39  } // end function setCommissionRate
40
41  // return commission rate
42  double CommissionEmployee::getCommissionRate() const
43  {
44     return commissionRate;
45  } // end function getCommissionRate
46
```

Fig. 12.14 | CommissionEmployee class implementation file. (Part 2 of 3.)

```
47  // calculate earnings; override pure virtual function earnings in Employee
48  double CommissionEmployee::earnings() const
49  {
50     return getCommissionRate() * getGrossSales();
51  } // end function earnings
52
53  // print CommissionEmployee's information
54  void CommissionEmployee::print() const
55  {
56     cout << "commission employee: ";
57     Employee::print(); // code reuse
58     cout << "\ngross sales: " << getGrossSales()
59        << "; commission rate: " << getCommissionRate();
60  } // end function print
```

Fig. 12.14 | `CommissionEmployee` class implementation file. (Part 3 of 3.)

12.6.4 Creating Indirect Concrete Derived Class `BasePlusCommissionEmployee`

- Class `BasePlusCommissionEmployee` (Figs. 12.15–12.16) directly inherits from class `CommissionEmployee` (line 9 of Fig. 12.15) and therefore is an indirect derived class of class `Employee`.
- `BasePlusCommissionEmployee`'s `print` function (lines 40–45) outputs `"base-salaried"`, followed by the output of base-class `CommissionEmployee`'s `print` function (another example of code reuse), then the base salary

```
1  // Fig. 12.15: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class derived from CommissionEmployee.
3  #ifndef BASEPLUS_H
4  #define BASEPLUS_H
5
6  #include <string> // C++ standard string class
7  #include "CommissionEmployee.h" // CommissionEmployee class definition
8
9  class BasePlusCommissionEmployee : public CommissionEmployee
10 {
11 public:
12    BasePlusCommissionEmployee( const std::string &, const std::string &,
13       const std::string &, double = 0.0, double = 0.0, double = 0.0 );
14    virtual ~CommissionEmployee() { } // virtual destructor
15
16    void setBaseSalary( double ); // set base salary
17    double getBaseSalary() const; // return base salary
18
```

Fig. 12.15 | BasePlusCommissionEmployee class header. (Part 1 of 2.)

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

Fig. 12.15 | BasePlusCommissionEmployee class header. (Part 2 of 2.)

```
1  // Fig. 12.16: BasePlusCommissionEmployee.cpp
2  // BasePlusCommissionEmployee member-function definitions.
3  #include <iostream>
4  #include <stdexcept>
5  #include "BasePlusCommissionEmployee.h"
6  using namespace std;
7
8  // constructor
9  BasePlusCommissionEmployee::BasePlusCommissionEmployee(
10    const string &first, const string &last, const string &ssn,
11    double sales, double rate, double salary )
12    : CommissionEmployee( first, last, ssn, sales, rate )
13 {
14    setBaseSalary( salary ); // validate and store base salary
15 } // end BasePlusCommissionEmployee constructor
16
```

Fig. 12.16 | BasePlusCommissionEmployee class implementation file. (Part 1 of 3.)

```
17  // set base salary
18  void BasePlusCommissionEmployee::setBaseSalary( double salary )
19  {
20     if ( salary >= 0.0 )
21        baseSalary = salary;
22     else
23        throw invalid_argument( "Salary must be >= 0.0" );
24  } // end function setBaseSalary
25
26  // return base salary
27  double BasePlusCommissionEmployee::getBaseSalary() const
28  {
29      return baseSalary;
30  } // end function getBaseSalary
31
32  // calculate earnings;
33  // override virtual function earnings in CommissionEmployee
34  double BasePlusCommissionEmployee::earnings() const
35  {
36      return getBaseSalary() + CommissionEmployee::earnings();
37  } // end function earnings
38
39  // print BasePlusCommissionEmployee's information
```

Fig. 12.16 | BasePlusCommissionEmployee class implementation file. (Part 2 of 3.)

```
40  void BasePlusCommissionEmployee::print() const
41  {
42     cout << "base-salaried ";
43     CommissionEmployee::print(); // code reuse
44     cout << "; base salary: " << getBaseSalary();
45  } // end function print
```

Fig. 12.16 | `BasePlusCommissionEmployee` class implementation file. (Part 3 of 3.)

12.6.5 Demonstrating Polymorphic Processing

- To test our `Employee` hierarchy, the program in Fig. 12.17 creates an object of each of the four concrete classes `SalariedEmployee`, `CommissionEmployee` and `BasePlusCommissionEmployee`.
- The program manipulates these objects, first with static binding, then polymorphically, using a `vector` of `Employee` pointers.
- Lines 22–27 create objects of each of the four concrete `Employee` derived classes.
- Lines 32–38 output each `Employee`'s information and earnings.
- Each member-function invocation in lines 32–37 is an example of *static binding*—at *compile time*, because we are using *name handles* (not *pointers* or *references* that could be set at *execution time*), the *compiler* can identify each object's type to determine which `print` and `earnings` functions are called.

```
1  // Fig. 12.17: fig12_17.cpp
2  // Processing Employee derived-class objects individually
3  // and polymorphically using dynamic binding.
4  #include <iostream>
5  #include <iomanip>
6  #include <vector>
7  #include "Employee.h"
8  #include "SalariedEmployee.h"
9  #include "CommissionEmployee.h"
10 #include "BasePlusCommissionEmployee.h"
11 using namespace std;
12
13 void virtualViaPointer( const Employee * const ); // prototype
14 void virtualViaReference( const Employee & ); // prototype
15
16 int main()
17 {
18    // set floating-point output formatting
19    cout << fixed << setprecision( 2 );
20
```

Fig. 12.17 | Employee class hierarchy driver program. (Part 1 of 7.)

```
21      // create derived-class objects
22      SalariedEmployee salariedEmployee(
23         "John", "Smith", "111-11-1111", 800 );
24      CommissionEmployee commissionEmployee(
25         "Sue", "Jones", "333-33-3333", 10000, .06 );
26      BasePlusCommissionEmployee basePlusCommissionEmployee(
27         "Bob", "Lewis", "444-44-4444", 5000, .04, 300 );
28
29      cout << "Employees processed individually using static binding:\n\n";
30
31      // output each Employee's information and earnings using static binding
32      salariedEmployee.print();
33      cout << "\nearned $" << salariedEmployee.earnings() << "\n\n";
34      commissionEmployee.print();
35      cout << "\nearned $" << commissionEmployee.earnings() << "\n\n";
36      basePlusCommissionEmployee.print();
37      cout << "\nearned $" << basePlusCommissionEmployee.earnings()
38         << "\n\n";
39
40      // create vector of three base-class pointers
41      vector< Employee * > employees( 3 );
42
```

Fig. 12.17 | Employee class hierarchy driver program. (Part 2 of 7.)

```
43      // initialize vector with pointers to Employees
44      employees[ 0 ] = &salariedEmployee;
45      employees[ 1 ] = &commissionEmployee;
46      employees[ 2 ] = &basePlusCommissionEmployee;
47
48      cout << "Employees processed polymorphically via dynamic binding:\n\n";
49
50      // call virtualViaPointer to print each Employee's information
51      // and earnings using dynamic binding
52      cout << "Virtual function calls made off base-class pointers:\n\n";
53
54      for ( const Employee *employeePtr : employees )
55         virtualViaPointer( employeePtr );
56
57      // call virtualViaReference to print each Employee's information
58      // and earnings using dynamic binding
59      cout << "Virtual function calls made off base-class references:\n\n";
60
61      for ( const Employee *employeePtr : employees )
62         virtualViaReference( *employeePtr ); // note dereferencing
63   } // end main
64
```

Fig. 12.17 | Employee class hierarchy driver program. (Part 3 of 7.)

```
65  // call Employee virtual functions print and earnings off a
66  // base-class pointer using dynamic binding
67  void virtualViaPointer( const Employee * const baseClassPtr )
68  {
69     baseClassPtr->print();
70     cout << "\nearned $" << baseClassPtr->earnings() << "\n\n";
71  } // end function virtualViaPointer
72
73  // call Employee virtual functions print and earnings off a
74  // base-class reference using dynamic binding
75  void virtualViaReference( const Employee &baseClassRef )
76  {
77     baseClassRef.print();
78     cout << "\nearned $" << baseClassRef.earnings() << "\n\n";
79  } // end function virtualViaReference
```

Fig. 12.17 | Employee class hierarchy driver program. (Part 4 of 7.)