Computer Programming
Chapter 13: Polymorphism - Part 2

Hung-Yu Wei
Department of Electrical Engineering
National Taiwan University

Example: Payroll System Using Polymorphism

- Section 13.6 (Fig13-Fig23)

- Base class (It’ s an abstract class)
  - *Employee*
  - Pure virtual function: *earnings*
- Derived class (They are concrete classes)
  - *SalariedEmployee*
  - *HourlyEmployee*
  - *CommissionEmployee*
  - *BasePlusCommissionEmployee*

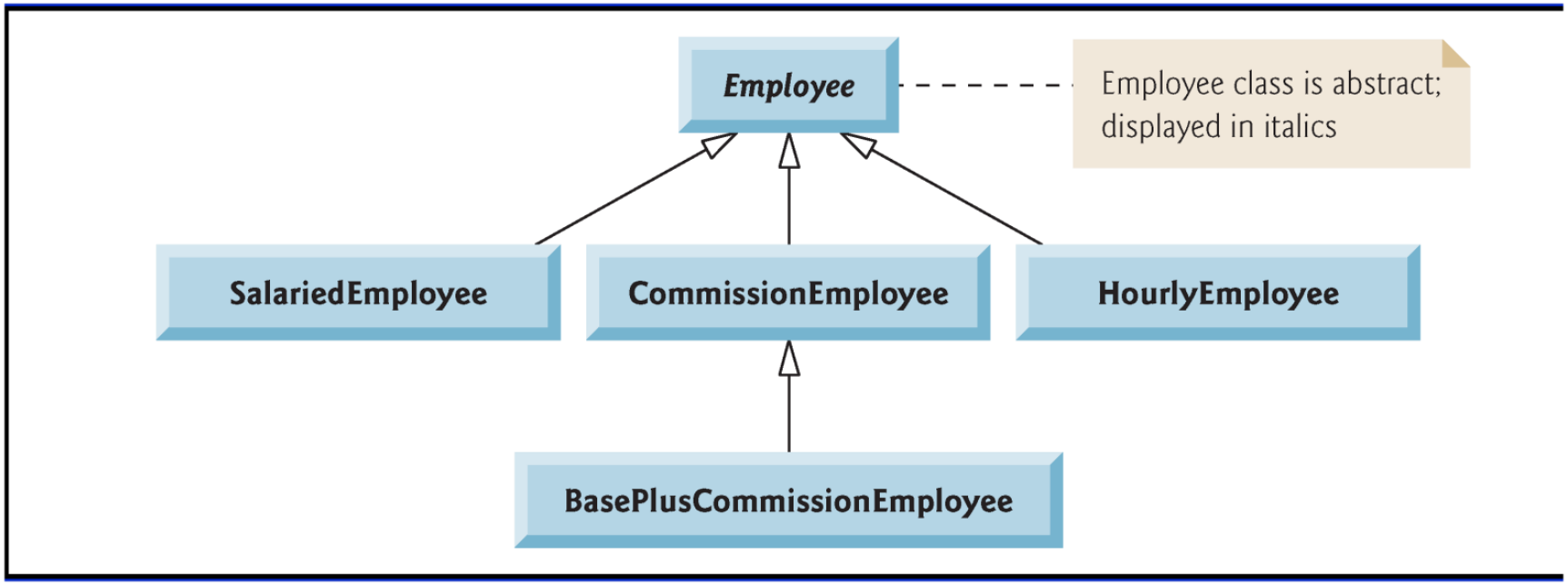


Fig. 13.11 | `Employee` hierarchy UML class diagram.

	earnings	print
Employee	= 0	*firstName lastName* social security number: *SSN*
Salaried-Employee	weeklySalary	salaried employee: *firstName lastName* social security number: *SSN* weekly salary: *weeklysalary*
Hourly-Employee	*If hours <= 40* wage * hours *If hours > 40* (40 * wage) + ((hours - 40) * wage * 1.5)	hourly employee: *firstName lastName* social security number: *SSN* hourly wage: *wage*; hours worked: *hours*
Commission-Employee	commissionRate * grossSales	commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales*; commission rate: *commissionRate*
BasePlus-Commission-Employee	baseSalary + (commissionRate * grossSales)	base salaried commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales*; commission rate: *commissionRate*; base salary: *baseSalary*

Fig. 13.12 | Polymorphic interface for the `Employee` hierarchy classes.

Vector (section 6.10)

- Default element values are 0
- Create a vector object with any data type
  - Syntax
    - vector <type> object_name(vector_size);
  - Examples
    - vector <int> MyVector1(5);
      - 5-element integer vector
    - vector <double> MyVector2(7);
      - 7-element double vector

Examples: Using vector objects

- myVector.size();
  - Size of the vector
- outputVector(myVector);
  - Show the data in the vector
- myVector1 = myVector2; // !=, ==, ...etc
  - Copy vector2 to vector1
- myVector[3] = 100;
  - Copy value 100 to element 3
  - Begin from element 0
- vector< int > myVectorNEW(myVectorOLD);
  - Declaration (copying the old vector to the new vector)
- vector< int > myVectorInteger(5);
  - Declaration a vector of 5 integers

```
1  // Fig. 13.13: Employee.h
2  // Employee abstract base class.
3  #ifndef EMPLOYEE_H
4  #define EMPLOYEE_H
5
6  #include <string> // C++ standard string class
7  using namespace std;
8
9  class Employee
10 {
11 public:
12    Employee( const string &, const string &, const string & );
13
14    void setFirstName( const string & ); // set first name
15    string getFirstName() const; // return first name
16
17    void setLastName( const string & ); // set last name
18    string getLastName() const; // return last name
19
20    void setSocialSecurityNumber( const string & ); // set SSN
21    string getSocialSecurityNumber() const; // return SSN
22
```

Fig. 13.13 | Employee class header file. (Part 1 of 2.)

```
23       // pure virtual function makes Employee abstract base class
24       virtual double earnings() const = 0; // pure virtual
25       virtual void print() const; // virtual
26    private:
27       string firstName;
28       string lastName;
29       string socialSecurityNumber;
30    }; // end class Employee
31
32    #endif // EMPLOYEE_H
```

Fig. 13.13 | `Employee` class header file. (Part 2 of 2.)

```
1  // Fig. 13.14: Employee.cpp
2  // Abstract-base-class Employee member-function definitions.
3  // Note: No definitions are given for pure virtual functions.
4  #include <iostream>
5  #include "Employee.h" // Employee class definition
6  using namespace std;
7
8  // constructor
9  Employee::Employee( const string &first, const string &last,
10    const string &ssn )
11    : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
12 {
13    // empty body
14 } // end Employee constructor
15
16 // set first name
17 void Employee::setFirstName( const string &first )
18 {
19    firstName = first;
20 } // end function setFirstName
21
```

Fig. 13.14 | Employee class implementation file. (Part 1 of 3.)

```
22  // return first name
23  string Employee::getFirstName() const
24  {
25     return firstName;
26  } // end function getFirstName
27
28  // set last name
29  void Employee::setLastName( const string &last )
30  {
31     lastName = last;
32  } // end function setLastName
33
34  // return last name
35  string Employee::getLastName() const
36  {
37     return lastName;
38  } // end function getLastName
39
40  // set social security number
41  void Employee::setSocialSecurityNumber( const string &ssn )
42  {
43     socialSecurityNumber = ssn; // should validate
44  } // end function setSocialSecurityNumber
45
```

Fig. 13.14 | `Employee` class implementation file. (Part 2 of 3.)

```
46  // return social security number
47  string Employee::getSocialSecurityNumber() const
48  {
49     return socialSecurityNumber;
50  } // end function getSocialSecurityNumber
51
52  // print Employee's information (virtual, but not pure virtual)
53  void Employee::print() const
54  {
55     cout << getFirstName() << ' ' << getLastName()
56        << "\nsocial security number: " << getSocialSecurityNumber();
57  } // end function print
```

Fig. 13.14 | `Employee` class implementation file. (Part 3 of 3.)

```
1  // Fig. 13.15: SalariedEmployee.h
2  // SalariedEmployee class derived from Employee.
3  #ifndef SALARIED_H
4  #define SALARIED_H
5
6  #include "Employee.h" // Employee class definition
7
8  class SalariedEmployee : public Employee
9  {
10 public:
11    SalariedEmployee( const string &, const string &,
12       const string &, double = 0.0 );
13
14    void setWeeklySalary( double ); // set weekly salary
15    double getWeeklySalary() const; // return weekly salary
16
17    // keyword virtual signals intent to override
18    virtual double earnings() const; // calculate earnings
19    virtual void print() const; // print SalariedEmployee object
20 private:
21    double weeklySalary; // salary per week
22 }; // end class SalariedEmployee
23
24 #endif // SALARIED_H
```

Fig. 13.15 | SalariedEmployee class header file.

```
1  // Fig. 13.16: SalariedEmployee.cpp
2  // SalariedEmployee class member-function definitions.
3  #include <iostream>
4  #include "SalariedEmployee.h" // SalariedEmployee class definition
5  using namespace std;
6
7  // constructor
8  SalariedEmployee::SalariedEmployee( const string &first,
9     const string &last, const string &ssn, double salary )
10    : Employee( first, last, ssn )
11 {
12    setWeeklySalary( salary );
13 } // end SalariedEmployee constructor
14
15 // set salary
16 void SalariedEmployee::setWeeklySalary( double salary )
17 {
18    weeklySalary = ( salary < 0.0 ) ? 0.0 : salary;
19 } // end function setWeeklySalary
20
```

Fig. 13.16 | `SalariedEmployee` class implementation file. (Part 1 of 2.)

```
21   // return salary
22   double SalariedEmployee::getWeeklySalary() const
23   {
24      return weeklySalary;
25   } // end function getWeeklySalary
26
27   // calculate earnings;
28   // override pure virtual function earnings in Employee
29   double SalariedEmployee::earnings() const
30   {
31      return getWeeklySalary();
32   } // end function earnings
33
34   // print SalariedEmployee's information
35   void SalariedEmployee::print() const
36   {
37      cout << "salaried employee: ";
38      Employee::print(); // reuse abstract base-class print function
39      cout << "\nweekly salary: " << getWeeklySalary();
40   } // end function print
```

Fig. 13.16 | `SalariedEmployee` class implementation file. (Part 2 of 2.)

```
1  // Fig. 13.17: HourlyEmployee.h
2  // HourlyEmployee class definition.
3  #ifndef HOURLY_H
4  #define HOURLY_H
5
6  #include "Employee.h" // Employee class definition
7
8  class HourlyEmployee : public Employee
9  {
10 public:
11    static const int hoursPerWeek = 168; // hours in one week
12
13    HourlyEmployee( const string &, const string &,
14       const string &, double = 0.0, double = 0.0 );
15
16    void setWage( double ); // set hourly wage
17    double getWage() const; // return hourly wage
18
19    void setHours( double ); // set hours worked
20    double getHours() const; // return hours worked
21
22    // keyword virtual signals intent to override
23    virtual double earnings() const; // calculate earnings
24    virtual void print() const; // print HourlyEmployee object
```

Fig. 13.17 | `HourlyEmployee` class header file. (Part 1 of 2.)

```
25   private:
26      double wage; // wage per hour
27      double hours; // hours worked for week
28   }; // end class HourlyEmployee
29
30   #endif // HOURLY_H
```

Fig. 13.17 | `HourlyEmployee` class header file. (Part 2 of 2.)

```
1  // Fig. 13.18: HourlyEmployee.cpp
2  // HourlyEmployee class member-function definitions.
3  #include <iostream>
4  #include "HourlyEmployee.h" // HourlyEmployee class definition
5  using namespace std;
6
7  // constructor
8  HourlyEmployee::HourlyEmployee( const string &first, const string &last,
9     const string &ssn, double hourlyWage, double hoursWorked )
10    : Employee( first, last, ssn )
11 {
12    setWage( hourlyWage ); // validate hourly wage
13    setHours( hoursWorked ); // validate hours worked
14 } // end HourlyEmployee constructor
15
16 // set wage
17 void HourlyEmployee::setWage( double hourlyWage )
18 {
19    wage = ( hourlyWage < 0.0 ? 0.0 : hourlyWage );
20 } // end function setWage
21
```

Fig. 13.18 | `HourlyEmployee` class implementation file. (Part 1 of 3.)

```
22   // return wage
23   double HourlyEmployee::getWage() const
24   {
25      return wage;
26   } // end function getWage
27
28   // set hours worked
29   void HourlyEmployee::setHours( double hoursWorked )
30   {
31      hours = ( ( ( hoursWorked >= 0.0 ) &&
32         ( hoursWorked <= hoursPerWeek ) ) ? hoursWorked : 0.0 );
33   } // end function setHours
34
35   // return hours worked
36   double HourlyEmployee::getHours() const
37   {
38      return hours;
39   } // end function getHours
40
```

Fig. 13.18 | `HourlyEmployee` class implementation file. (Part 2 of 3.)

```
41  // calculate earnings;
42  // override pure virtual function earnings in Employee
43  double HourlyEmployee::earnings() const
44  {
45     if ( getHours() <= 40 ) // no overtime
46        return getWage() * getHours();
47     else
48        return 40 * getWage() + ( ( getHours() - 40 ) * getWage() * 1.5 );
49  } // end function earnings
50
51  // print HourlyEmployee's information
52  void HourlyEmployee::print() const
53  {
54     cout << "hourly employee: ";
55     Employee::print(); // code reuse
56     cout << "\nhourly wage: " << getWage() <<
57        "; hours worked: " << getHours();
58  } // end function print
```

Fig. 13.18 | `HourlyEmployee` class implementation file. (Part 3 of 3.)

```
1  // Fig. 13.19: CommissionEmployee.h
2  // CommissionEmployee class derived from Employee.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include "Employee.h" // Employee class definition
7
8  class CommissionEmployee : public Employee
9  {
10 public:
11    CommissionEmployee( const string &, const string &,
12       const string &, double = 0.0, double = 0.0 );
13
14    void setCommissionRate( double ); // set commission rate
15    double getCommissionRate() const; // return commission rate
16
17    void setGrossSales( double ); // set gross sales amount
18    double getGrossSales() const; // return gross sales amount
19
20    // keyword virtual signals intent to override
21    virtual double earnings() const; // calculate earnings
22    virtual void print() const; // print CommissionEmployee object
```

Fig. 13.19 | `CommissionEmployee` class header file. (Part 1 of 2.)

```
23   private:
24      double grossSales; // gross weekly sales
25      double commissionRate; // commission percentage
26   }; // end class CommissionEmployee
27
28   #endif // COMMISSION_H
```

Fig. 13.19 | `CommissionEmployee` class header file. (Part 2 of 2.)

```
1  // Fig. 13.20: CommissionEmployee.cpp
2  // CommissionEmployee class member-function definitions.
3  #include <iostream>
4  #include "CommissionEmployee.h" // CommissionEmployee class definition
5  using namespace std;
6
7  // constructor
8  CommissionEmployee::CommissionEmployee( const string &first,
9     const string &last, const string &ssn, double sales, double rate )
10    : Employee( first, last, ssn )
11 {
12    setGrossSales( sales );
13    setCommissionRate( rate );
14 } // end CommissionEmployee constructor
15
16 // set commission rate
17 void CommissionEmployee::setCommissionRate( double rate )
18 {
19    commissionRate = ( ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0 );
20 } // end function setCommissionRate
21
```

Fig. 13.20 | `CommissionEmployee` class implementation file. (Part 1 of 3.)

```
22  // return commission rate
23  double CommissionEmployee::getCommissionRate() const
24  {
25      return commissionRate;
26  } // end function getCommissionRate
27
28  // set gross sales amount
29  void CommissionEmployee::setGrossSales( double sales )
30  {
31     grossSales = ( ( sales < 0.0 ) ? 0.0 : sales );
32  } // end function setGrossSales
33
34  // return gross sales amount
35  double CommissionEmployee::getGrossSales() const
36  {
37      return grossSales;
38  } // end function getGrossSales
39
40  // calculate earnings; override pure virtual function earnings in Employee
41  double CommissionEmployee::earnings() const
42  {
43     return getCommissionRate() * getGrossSales();
44  } // end function earnings
45
```

Fig. 13.20 | `CommissionEmployee` class implementation file. (Part 2 of 3.)

```
46  // print CommissionEmployee's information
47  void CommissionEmployee::print() const
48  {
49     cout << "commission employee: ";
50     Employee::print(); // code reuse
51     cout << "\ngross sales: " << getGrossSales()
52        << "; commission rate: " << getCommissionRate();
53  } // end function print
```

Fig. 13.20 | `CommissionEmployee` class implementation file. (Part 3 of 3.)

```
1  // Fig. 13.21: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class derived from CommissionEmployee.
3  #ifndef BASEPLUS_H
4  #define BASEPLUS_H
5
6  #include "CommissionEmployee.h" // CommissionEmployee class definition
7
8  class BasePlusCommissionEmployee : public CommissionEmployee
9  {
10 public:
11    BasePlusCommissionEmployee( const string &, const string &,
12       const string &, double = 0.0, double = 0.0, double = 0.0 );
13
14    void setBaseSalary( double ); // set base salary
15    double getBaseSalary() const; // return base salary
16
17    // keyword virtual signals intent to override
18    virtual double earnings() const; // calculate earnings
19    virtual void print() const; // print BasePlusCommissionEmployee object
20 private:
21    double baseSalary; // base salary per week
22 }; // end class BasePlusCommissionEmployee
23
24 #endif // BASEPLUS_H
```

Fig. 13.21 | BasePlusCommissionEmployee class header file.

```
1  // Fig. 13.22: BasePlusCommissionEmployee.cpp
2  // BasePlusCommissionEmployee member-function definitions.
3  #include <iostream>
4  #include "BasePlusCommissionEmployee.h"
5  using namespace std;
6
7  // constructor
8  BasePlusCommissionEmployee::BasePlusCommissionEmployee(
9     const string &first, const string &last, const string &ssn,
10    double sales, double rate, double salary )
11    : CommissionEmployee( first, last, ssn, sales, rate )
12 {
13    setBaseSalary( salary ); // validate and store base salary
14 } // end BasePlusCommissionEmployee constructor
15
16 // set base salary
17 void BasePlusCommissionEmployee::setBaseSalary( double salary )
18 {
19    baseSalary = ( ( salary < 0.0 ) ? 0.0 : salary );
20 } // end function setBaseSalary
21
```

Fig. 13.22 | `BasePlusCommissionEmployee` class implementation file. (Part 1 of 2.)

```
22  // return base salary
23  double BasePlusCommissionEmployee::getBaseSalary() const
24  {
25      return baseSalary;
26  } // end function getBaseSalary
27
28  // calculate earnings;
29  // override virtual function earnings in CommissionEmployee
30  double BasePlusCommissionEmployee::earnings() const
31  {
32      return getBaseSalary() + CommissionEmployee::earnings();
33  } // end function earnings
34
35  // print BasePlusCommissionEmployee's information
36  void BasePlusCommissionEmployee::print() const
37  {
38     cout << "base-salaried ";
39     CommissionEmployee::print(); // code reuse
40     cout << "; base salary: " << getBaseSalary();
41  } // end function print
```

Fig. 13.22 | `BasePlusCommissionEmployee` class implementation file. (Part 2 of 2.)

```
1  // Fig. 13.23: fig13_23.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 "HourlyEmployee.h"
10 #include "CommissionEmployee.h"
11 #include "BasePlusCommissionEmployee.h"
12 using namespace std;
13
14 void virtualViaPointer( const Employee * const ); // prototype
15 void virtualViaReference( const Employee & ); // prototype
16
17 int main()
18 {
19    // set floating-point output formatting
20    cout << fixed << setprecision( 2 );
21
```

Fig. 13.23 | `Employee` class hierarchy driver program. (Part 1 of 7.)

```
22      // create derived-class objects
23      SalariedEmployee salariedEmployee(
24         "John", "Smith", "111-11-1111", 800 );
25      HourlyEmployee hourlyEmployee(
26         "Karen", "Price", "222-22-2222", 16.75, 40 );
27      CommissionEmployee commissionEmployee(
28         "Sue", "Jones", "333-33-3333", 10000, .06 );
29      BasePlusCommissionEmployee basePlusCommissionEmployee(
30         "Bob", "Lewis", "444-44-4444", 5000, .04, 300 );
31
32      cout << "Employees processed individually using static binding:\n\n";
33
34      // output each Employee's information and earnings using static binding
35      salariedEmployee.print();
36      cout << "\nearned $" << salariedEmployee.earnings() << "\n\n";
37      hourlyEmployee.print();
38      cout << "\nearned $" << hourlyEmployee.earnings() << "\n\n";
39      commissionEmployee.print();
40      cout << "\nearned $" << commissionEmployee.earnings() << "\n\n";
41      basePlusCommissionEmployee.print();
42      cout << "\nearned $" << basePlusCommissionEmployee.earnings()
43         << "\n\n";
44
```

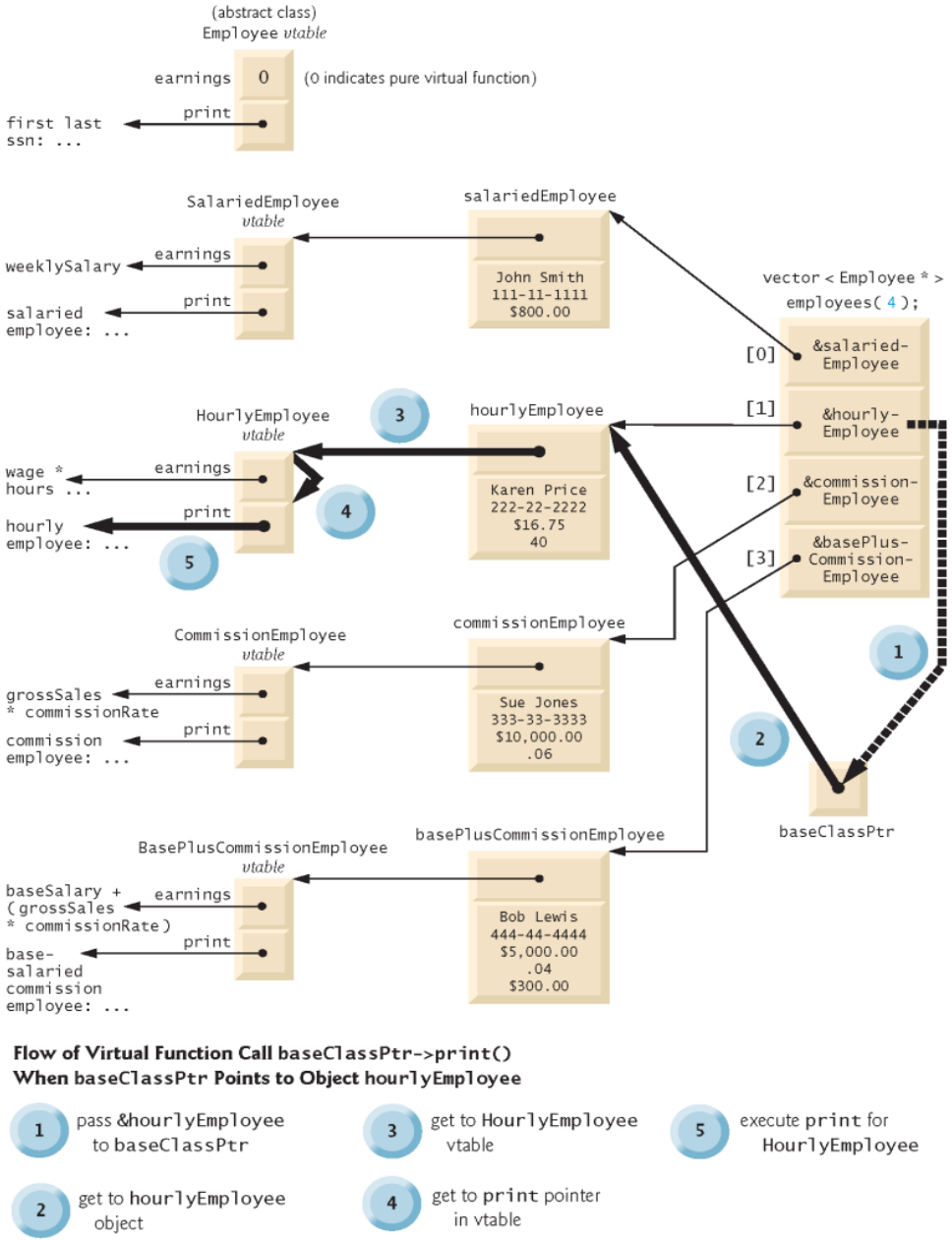
Fig. 13.23 | Employee class hierarchy driver program. (Part 2 of 7.)

```
45      // create vector of four base-class pointers
46      vector < Employee * > employees( 4 );
47
48      // initialize vector with Employees
49      employees[ 0 ] = &salariedEmployee;
50      employees[ 1 ] = &hourlyEmployee;
51      employees[ 2 ] = &commissionEmployee;
52      employees[ 3 ] = &basePlusCommissionEmployee;
53
54      cout << "Employees processed polymorphically via dynamic binding:\n\n";
55
56      // call virtualViaPointer to print each Employee's information
57      // and earnings using dynamic binding
58      cout << "Virtual function calls made off base-class pointers:\n\n";
59
60      for ( size_t i = 0; i < employees.size(); i++ )
61         virtualViaPointer( employees[ i ] );
62
63      // call virtualViaReference to print each Employee's information
64      // and earnings using dynamic binding
65      cout << "Virtual function calls made off base-class references:\n\n";
66
67      for ( size_t i = 0; i < employees.size(); i++ )
68         virtualViaReference( *employees[ i ] ); // note dereferencing
69   } // end main
```

Fig. 13.23 | `Employee` class hierarchy driver program. (Part 3 of 7.)

```
70
71    // call Employee virtual functions print and earnings off a
72    // base-class pointer using dynamic binding
73    void virtualViaPointer( const Employee * const baseClassPtr )
74    {
75       baseClassPtr->print();
76       cout << "\nearned $" << baseClassPtr->earnings() << "\n\n";
77    } // end function virtualViaPointer
78
79    // call Employee virtual functions print and earnings off a
80    // base-class reference using dynamic binding
81    void virtualViaReference( const Employee &baseClassRef )
82    {
83       baseClassRef.print();
84       cout << "\nearned $" << baseClassRef.earnings() << "\n\n";
85    } // end function virtualViaReference
```

Fig. 13.23 | `Employee` class hierarchy driver program. (Part 4 of 7.)



Flow of Virtual Function Call `baseClassPtr->print()`
When `baseClassPtr` Points to Object `hourlyEmployee`

1. pass `&hourlyEmployee` to `baseClassPtr`
2. get to `hourlyEmployee` object
3. get to `HourlyEmployee` vtable
4. get to `print` pointer in vtable
5. execute `print` for `HourlyEmployee`

```
baseClassPtr->print();
```

Fig. 13.24 | How `virtual` function calls work.

vector

- Member function *size*
  - Return the number of elements
- Operations on vector objects
  - !=
  - ==
  - =
- Access or modify an element in vector
  - []
- Bound checking

Section 13.8

- Example: Fig 13.25
  - Reward BasePlusCommissionEmployees by adding 10 percent to their base salaries.

- Concepts and syntax
  - Downcast
  - Dynamic_cast
  - Type_info
  - runtime type information (RTTI)

Type_info

- # include <typeinfo>

- Get the string that contains the type name

```
cout << typeid( *MyEmployee).name();
```

Dynamic casting

- **Dynamic_cast**
  - Convert the type of object
    - Object to be converted
    - new type

```
dynamic_cast < BasePlusCommissionEmployee * > (
```

  - When the casted type is a pointer
    - If invalid
      - Return Null pointer
    - If valid
      - pointer to the requested object

[concept] downcast

- Error: cannot assign a base-class pointer to a derived class pointer

- Solution
  - Use dynamic_cast to avoid error

- Downcast
  - convert to derived class

```
1  // Fig. 13.25: fig13_25.cpp
2  // Demonstrating downcasting and runtime type information.
3  // NOTE: You may need to enable RTTI on your compiler
4  // before you can execute this application.
5  #include <iostream>
6  #include <iomanip>
7  #include <vector>
8  #include <typeinfo>
9  #include "Employee.h"
10 #include "SalariedEmployee.h"
11 #include "HourlyEmployee.h"
12 #include "CommissionEmployee.h"
13 #include "BasePlusCommissionEmployee.h"
14 using namespace std;
15
16 int main()
17 {
18    // set floating-point output formatting
19    cout << fixed << setprecision( 2 );
20
21    // create vector of four base-class pointers
22    vector < Employee * > employees( 4 );
23
```

Fig. 13.25 | Demonstrating downcasting and runtime type information. (Part 1 of 4.)

```
24       // initialize vector with various kinds of Employees
25       employees[ 0 ] = new SalariedEmployee(
26          "John", "Smith", "111-11-1111", 800 );
27       employees[ 1 ] = new HourlyEmployee(
28          "Karen", "Price", "222-22-2222", 16.75, 40 );
29       employees[ 2 ] = new CommissionEmployee(
30          "Sue", "Jones", "333-33-3333", 10000, .06 );
31       employees[ 3 ] = new BasePlusCommissionEmployee(
32          "Bob", "Lewis", "444-44-4444", 5000, .04, 300 );
33
34       // polymorphically process each element in vector employees
35       for ( size_t i = 0; i < employees.size(); i++ )
36       {
37          employees[ i ]->print(); // output employee information
38          cout << endl;
39
40          // downcast pointer
41          BasePlusCommissionEmployee *derivedPtr =
42             dynamic_cast < BasePlusCommissionEmployee * >
43                ( employees[ i ] );
44
```

Fig. 13.25 | Demonstrating downcasting and runtime type information. (Part 2 of 4.)

```
45          // determine whether element points to base-salaried
46          // commission employee
47          if ( derivedPtr != 0 ) // 0 if not a BasePlusCommissionEmployee
48          {
49             double oldBaseSalary = derivedPtr->getBaseSalary();
50             cout << "old base salary: $" << oldBaseSalary << endl;
51             derivedPtr->setBaseSalary( 1.10 * oldBaseSalary );
52             cout << "new base salary with 10% increase is: $"
53                << derivedPtr->getBaseSalary() << endl;
54          } // end if
55
56          cout << "earned $" << employees[ i ]->earnings() << "\n\n";
57       } // end for
58
59       // release objects pointed to by vector's elements
60       for ( size_t j = 0; j < employees.size(); j++ )
61       {
62          // output class name
63          cout << "deleting object of "
64             << typeid( *employees[ j ] ).name() << endl;
65
66          delete employees[ j ];
67       } // end for
68    } // end main
```

Fig. 13.25 | Demonstrating downcasting and runtime type information. (Part 3 of 4.)

```
salaried employee: John Smith
social security number: 111-11-1111
weekly salary: 800.00
earned $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: 16.75; hours worked: 40.00
earned $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: 10000.00; commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: 5000.00; commission rate: 0.04; base salary: 300.00
old base salary: $300.00
new base salary with 10% increase is: $330.00
earned $530.00

deleting object of class SalariedEmployee
deleting object of class HourlyEmployee
deleting object of class CommissionEmployee
deleting object of class BasePlusCommissionEmployee
```

Fig. 13.25 | Demonstrating downcasting and runtime type information. (Part 4 of 4.)

Section 13.5 Abstract Class and Pure Virtual Function

- Pure virtual function
  - Provide no implementation
  - Require derived class to override the base-class pure virtual function
  - Pure specifier =0

    ```
    virtual void draw() =0;
    ```
- Abstract class
  - We will **NOT** use this class to create objects
  - Typically use as base class for inheritance
  - Include 1 or more pure virtual function
- Concrete class
  - 相反的概念: abstract class
  - Class that can be used to instantiate objects

13.9 virtual destructor

- Virtual destructor
  - A destructor with "virtual" keyword
- Use virtual destructor in base-class
  - All derived class destructors are all virtual
  - When use *delete* operation on base-class pointer
    - The corresponding destructor will be called
- Constructor cannot be virtual

Summary: Polymorphism

- Polymorphism
- Base-class pointer
- Virtual function
- Abstract class
  - Pure virtual function
  - Concrete class