Computer Programming
Chapter 13: Polymorphism - Part 1

Hung-Yu Wei
Department of Electrical Engineering
National Taiwan University

Chapter 13: Polymorphism

- Polymorphism
  - Makes program more general and extensible
  - A function can cause different actions, depending on which function is called
- Abstract class
  - Abstract base class
  - Pointer to the base class
- virtual function
  - Virtual function could be re-defined in derived class

[concept]Pointers to base class

- A pointer to a derived class is type-compatible with a pointer to its base class
  - Polymorphism utilizes this feature

```
1  // pointers to base class
2  #include <iostream>
3  using namespace std;
4
5  class CPolygon {
6    protected:
7      int width, height;
8    public:
9      void set_values (int a, int b)
10       { width=a; height=b; }
11   };
```

Base Class

Derived Class

```
13 class CRectangle: public CPolygon {
14   public:
15     int area ()
16       { return (width * height); }
17   };
18
19 class CRightTriangle: public CPolygon {
20   public:
21     int area ()
22       { return (width * height / 2); }
23   };
```

```
25 int main () {
26   CRectangle rect;
27   CRightTriangle trgl;
28   CPolygon * ppoly1 = &rect;
29   CPolygon * ppoly2 = &trgl;
30   ppoly1->set_values (4,5);
31   ppoly2->set_values (4,5);
32
33   cout << rect.area() << endl;
34   //  cout << ppoly1->area() << endl;  //error
35   cout << trgl.area() << endl;
36 }
```

Pointer to base class

[concept]Virtual member

- Virtual member
  - A member of a class that can be redefined in its derived classes
- Syntax

  virtual OutType myFunction (InType)

- Example
  - In base class CPolygon

```
virtual int area ()
  { return (0); }
```

  - In derived class
    - In CRectangle

```
int area ()
   { return (width * height); }
```

    - In CTriangle

```
int area ()
    { return (width * height / 2);
}
```

```
1  // virtual members
2  #include <iostream>
3  using namespace std;
4
5  class CPolygon {
6    protected:
7      int width, height;
8    public:
9      void set_values (int a, int b)
10       { width=a; height=b; }
11
12     virtual int area ()
13       { return (0); }
14   };
```

Base Class

```
16 class CRectangle: public CPolygon {
17   public:
18     int area ()
19       { return (width * height); }
20   };
21
22 class CTriangle: public CPolygon {
23   public:
24     int area ()
25       { return (width * height / 2); }
26   };
```

Derived Class

```
28 int main () {
29   CRectangle rect;
30   CTriangle trgl;
31   CPolygon poly;
32   CPolygon * ppoly1 = &rect;
33   CPolygon * ppoly2 = &trgl;
34   CPolygon * ppoly3 = &poly;
35   ppoly1->set_values (4,5);
36   ppoly2->set_values (4,5);
37   ppoly3->set_values (4,5);
38   cout << ppoly1->area() << endl;
39   cout << ppoly2->area() << endl;
40   cout << ppoly3->area() << endl;
41 }
```

[concept] Abstract base classes

- Virtual class without implementation

- Example

```
virtual int area () =0;
```

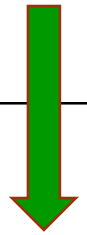
- Cannot create base class object
  - Because virtual function is abstract

```
CPolygon poly1;                    //ERROR
```

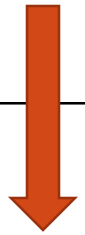
- Can be used to create pointer to base class object

```
CPolygon * PTRpoly2;     //OK
```

```
1  // abstract base class
2  #include <iostream>
3  using namespace std;
4
5  class CPolygon {
6    protected:
7      int width, height;
8    public:
9      void set_values (int a, int b)
10       { width=a; height=b; }
11     virtual int area (void) =0;
12   };
```



Virtual



Abstract (No implementation)

```
14 class CRectangle: public CPolygon {
15   public:
16     int area (void)
17       { return (width * height); }
18   };
19
20 class CTriangle: public CPolygon {
21   public:
22     int area (void)
23       { return (width * height / 2); }
24   };
```

```
26 int main () {
27   CRectangle rect;
28   CTriangle trgl;
29   CPolygon * ppoly1 = &rect;
30   CPolygon * ppoly2 = &trgl;
31   ppoly1->set_values (4,5);
32   ppoly2->set_values (4,5);
33   cout << ppoly1->area() << endl;
34   cout << ppoly2->area() << endl;
35   return 0;
36 }
```

Example: virtual function and abstract base class

```
1  // pure virtual members can be called
2  // from the abstract base class
3  #include <iostream>
4  using namespace std;
5
6  class CPolygon {
7    protected:
8      int width, height;
9    public:
10     void set_values (int a, int b)
11       { width=a; height=b; }
12     virtual int area (void) =0;
13     void printarea (void)
14       { cout << this->area() << endl; }
15   };
```

```
17 class CRectangle: public CPolygon {
18   public:
19     int area (void)
20       { return (width * height); }
21   };
22
23 class CTriangle: public CPolygon {
24   public:
25     int area (void)
26       { return (width * height / 2); }
27   };
```

```
29 int main () {
30   CRectangle rect;
31   CTriangle trgl;
32   CPolygon * ppoly1 = &rect;
33   CPolygon * ppoly2 = &trgl;
34   ppoly1->set_values (4,5);
35   ppoly2->set_values (4,5);
36   ppoly1->printarea();
37   ppoly2->printarea();
38   return 0;
39 }
```

Example: dynamic allocation and polymorphism

- Dynamic allocation using *new* and *delete*

- new

```
CPolygon * ppoly1 = new CRectangle;
CPolygon * ppoly2 = new CTriangle;
```

- delete

```
delete ppoly1;
delete ppoly2;
```

```
1  // dynamic allocation and polymorphism
2  #include <iostream>
3  using namespace std;
4
5  class CPolygon {
6    protected:
7      int width, height;
8    public:
9      void set_values (int a, int b)
10       { width=a; height=b; }
11     virtual int area (void) =0;
12     void printarea (void)
13       { cout << this->area() << endl; }
14   };
```

```
16 class CRectangle: public CPolygon {
17   public:
18     int area (void)
19       { return (width * height); }
20   };
21
22 class CTriangle: public CPolygon {
23   public:
24     int area (void)
25       { return (width * height / 2); }
26   };
```

```
28 int main () {
29   CPolygon * ppoly1 = new CRectangle;
30   CPolygon * ppoly2 = new CTriangle;
31   ppoly1->set_values (4,5);
32   ppoly2->set_values (4,5);
33   ppoly1->printarea();
34   ppoly2->printarea();
35   delete ppoly1;
36   delete ppoly2;
37   return 0;
38 }
```

Textbook Examples

Textbook Examples in 13.3

- Aim base-class and derived-class pointers to base-class and derived-class objects
  - 13.3.1 (Fig13.1-Fig13.5)
  - 13.3.2 Error case (Fig13.6)
    - derived class pointer aims at base class object
  - 13.3.3 Error case (Fig13.7)
    - base-class pointer calls derived-class-only member function
  - 13.3.4 Virtual Function (Fig 13.8-13.10)
- Scenario (extend example in Chapter 12)
  - Base class: CommissionEmployee
  - Derived class: BasePlusCommissionEmployee

13.3.1: Invoking base-class function from derived-class objects

Example: aim pointer to object

- Object of
  - Base class
  - Derived class
- Pointer of
  - Base class
  - Derived class
- 4 cases
  - OK (Section 13.3.1)
    - aim base-class pointer at base-class object
    - aim derived-class pointer at derived-class object
    - aim base-class pointer at derived-class object
  - Error(Section 13.3.2)
    - aim derived-class pointer at base-class object

```
1  // Fig. 13.1: CommissionEmployee.h
2  // CommissionEmployee class definition represents a commission employee.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // C++ standard string class
7  using namespace std;
8
9  class CommissionEmployee
10 {
11 public:
12    CommissionEmployee( const string &, const string &, const string &,
13       double = 0.0, double = 0.0 );
14
15    void setFirstName( const string & ); // set first name
16    string getFirstName() const; // return first name
17
18    void setLastName( const string & ); // set last name
19    string getLastName() const; // return last name
20
21    void setSocialSecurityNumber( const string & ); // set SSN
22    string getSocialSecurityNumber() const; // return SSN
23
```

Fig. 13.1 | CommissionEmployee class header file. (Part 1 of 2.)

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

Fig. 13.1 | `CommissionEmployee` class header file. (Part 2 of 2.)

```
1  // Fig. 13.2: CommissionEmployee.cpp
2  // Class CommissionEmployee member-function definitions.
3  #include <iostream>
4  #include "CommissionEmployee.h" // CommissionEmployee class definition
5  using namespace std;
6
7  // constructor
8  CommissionEmployee::CommissionEmployee(
9     const string &first, const string &last, const string &ssn,
10    double sales, double rate )
11    : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
12 {
13    setGrossSales( sales ); // validate and store gross sales
14    setCommissionRate( rate ); // validate and store commission rate
15 } // end CommissionEmployee constructor
16
17 // set first name
18 void CommissionEmployee::setFirstName( const string &first )
19 {
20    firstName = first; // should validate
21 } // end function setFirstName
22
```

Fig. 13.2 | `CommissionEmployee` class implementation file. (Part 1 of 4.)

```
23  // return first name
24  string CommissionEmployee::getFirstName() const
25  {
26     return firstName;
27  } // end function getFirstName
28
29  // set last name
30  void CommissionEmployee::setLastName( const string &last )
31  {
32     lastName = last;  // should validate
33  } // end function setLastName
34
35  // return last name
36  string CommissionEmployee::getLastName() const
37  {
38     return lastName;
39  } // end function getLastName
40
41  // set social security number
42  void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
43  {
44     socialSecurityNumber = ssn; // should validate
45  } // end function setSocialSecurityNumber
46
```

Fig. 13.2 | `CommissionEmployee` class implementation file. (Part 2 of 4.)

```
47  // return social security number
48  string CommissionEmployee::getSocialSecurityNumber() const
49  {
50     return socialSecurityNumber;
51  } // end function getSocialSecurityNumber
52
53  // set gross sales amount
54  void CommissionEmployee::setGrossSales( double sales )
55  {
56     grossSales = ( sales < 0.0 ) ? 0.0 : sales;
57  } // end function setGrossSales
58
59  // return gross sales amount
60  double CommissionEmployee::getGrossSales() const
61  {
62     return grossSales;
63  } // end function getGrossSales
64
65  // set commission rate
66  void CommissionEmployee::setCommissionRate( double rate )
67  {
68     commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
69  } // end function setCommissionRate
70
```

Fig. 13.2 | `CommissionEmployee` class implementation file. (Part 3 of 4.)

```
71  // return commission rate
72  double CommissionEmployee::getCommissionRate() const
73  {
74     return commissionRate;
75  } // end function getCommissionRate
76
77  // calculate earnings
78  double CommissionEmployee::earnings() const
79  {
80     return getCommissionRate() * getGrossSales();
81  } // end function earnings
82
83  // print CommissionEmployee object
84  void CommissionEmployee::print() const
85  {
86     cout << "commission employee: "
87        << getFirstName() << ' ' << getLastName()
88        << "\nsocial security number: " << getSocialSecurityNumber()
89        << "\ngross sales: " << getGrossSales()
90        << "\ncommission rate: " << getCommissionRate();
91  } // end function print
```

Fig. 13.2 | `CommissionEmployee` class implementation file. (Part 4 of 4.)

```
1  // Fig. 13.3: 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  using namespace std;
10
11 class BasePlusCommissionEmployee : public CommissionEmployee
12 {
13 public:
14    BasePlusCommissionEmployee( const string &, const string &,
15       const string &, double = 0.0, double = 0.0, double = 0.0 );
16
17    void setBaseSalary( double ); // set base salary
18    double getBaseSalary() const; // return base salary
19
20    double earnings() const; // calculate earnings
21    void print() const; // print BasePlusCommissionEmployee object
```

Fig. 13.3 | BasePlusCommissionEmployee class header file. (Part 1 of 2.)

```
22   private:
23      double baseSalary; // base salary
24   }; // end class BasePlusCommissionEmployee
25
26   #endif
```

Fig. 13.3 | `BasePlusCommissionEmployee` class header file. (Part 2 of 2.)

```
1  // Fig. 13.4: BasePlusCommissionEmployee.cpp
2  // Class 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    // explicitly call base-class constructor
12    : CommissionEmployee( first, last, ssn, sales, rate )
13 {
14    setBaseSalary( salary ); // validate and store base salary
15 } // end BasePlusCommissionEmployee constructor
16
17 // set base salary
18 void BasePlusCommissionEmployee::setBaseSalary( double salary )
19 {
20    baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
21 } // end function setBaseSalary
22
```

Fig. 13.4 | `BasePlusCommissionEmployee` class implementation file. (Part 1 of 2.)

```
23  // return base salary
24  double BasePlusCommissionEmployee::getBaseSalary() const
25  {
26     return baseSalary;
27  } // end function getBaseSalary
28
29  // calculate earnings
30  double BasePlusCommissionEmployee::earnings() const
31  {
32     return getBaseSalary() + CommissionEmployee::earnings();
33  } // end function earnings
34
35  // print BasePlusCommissionEmployee object
36  void BasePlusCommissionEmployee::print() const
37  {
38     cout << "base-salaried ";
39
40     // invoke CommissionEmployee's print function
41     CommissionEmployee::print();
42
43     cout << "\nbase salary: " << getBaseSalary();
44  } // end function print
```

Fig. 13.4 | `BasePlusCommissionEmployee` class implementation file. (Part 2 of 2.)

```
1  // Fig. 13.5: fig13_05.cpp
2  // Aiming base-class and derived-class pointers at base-class
3  // and derived-class objects, respectively.
4  #include <iostream>
5  #include <iomanip>
6  #include "CommissionEmployee.h"
7  #include "BasePlusCommissionEmployee.h"
8  using namespace std;
9
10 int main()
11 {
12    // create base-class object
13    CommissionEmployee commissionEmployee(
14       "Sue", "Jones", "222-22-2222", 10000, .06 );
15
16    // create base-class pointer
17    CommissionEmployee *commissionEmployeePtr = 0;
18
19    // create derived-class object
20    BasePlusCommissionEmployee basePlusCommissionEmployee(
21       "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
22
```

Fig. 13.5 | Assigning addresses of base-class and derived-class objects to base-class and derived-class pointers. (Part 1 of 5.)

```
23      // create derived-class pointer
24      BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = 0;
25
26      // set floating-point output formatting
27      cout << fixed << setprecision( 2 );
28
29      // output objects commissionEmployee and basePlusCommissionEmployee
30      cout << "Print base-class and derived-class objects:\n\n";
31      commissionEmployee.print(); // invokes base-class print
32      cout << "\n\n";
33      basePlusCommissionEmployee.print(); // invokes derived-class print
34
35      // aim base-class pointer at base-class object and print
36      commissionEmployeePtr = &commissionEmployee; // perfectly natural
37      cout << "\n\n\nCalling print with base-class pointer to "
38         << "\nbase-class object invokes base-class print function:\n\n";
39      commissionEmployeePtr->print(); // invokes base-class print
40
41      // aim derived-class pointer at derived-class object and print
42      basePlusCommissionEmployeePtr = &basePlusCommissionEmployee; // natural
43      cout << "\n\n\nCalling print with derived-class pointer to "
44         << "\nderived-class object invokes derived-class "
45         << "print function:\n\n";
```

Fig. 13.5 | Assigning addresses of base-class and derived-class objects to base-class and derived-class pointers. (Part 2 of 5.)

```
46      basePlusCommissionEmployeePtr->print(); // invokes derived-class print
47
48      // aim base-class pointer at derived-class object and print
49      commissionEmployeePtr = &basePlusCommissionEmployee;
50      cout << "\n\n\nCalling print with base-class pointer to "
51         << "derived-class object\ninvokes base-class print "
52         << "function on that derived-class object:\n\n";
53      commissionEmployeePtr->print(); // invokes base-class print
54      cout << endl;
55   } // end main
```

```
Print base-class and derived-class objects:

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

Fig. 13.5 | Assigning addresses of base-class and derived-class objects to base-class and derived-class pointers. (Part 3 of 5.)

13.3.2: Aiming Derived-Class Pointers at Base-Class Objects

Error: aim derived class pointer at base class object

```
1  // Fig. 13.6: fig13_06.cpp
2  // Aiming a derived-class pointer at a base-class object.
3  #include "CommissionEmployee.h"
4  #include "BasePlusCommissionEmployee.h"
5
6  int main()
7  {
8     CommissionEmployee commissionEmployee(
9        "Sue", "Jones", "222-22-2222", 10000, .06 );
10    BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = 0;
11
12    // aim derived-class pointer at base-class object
13    // Error: a CommissionEmployee is not a BasePlusCommissionEmployee
14    basePlusCommissionEmployeePtr = &commissionEmployee;
15 } // end main
```

Fig. 13.6 | Aiming a derived-class pointer at a base-class object. (Part 1 of 2.)

13.3.3:Derived-Class Member-Function Calls via Base-Class Pointers

Error: base-class pointer calls derived-class-only member function

- Base-class pointer is OK to aim at derived-class object

- But, base-class pointer **cannot** call the derived-class only member function
  - Exception: use *downcasting* technique (Section 13.8)

```
1  // Fig. 13.7: fig13_07.cpp
2  // Attempting to invoke derived-class-only member functions
3  // through a base-class pointer.
4  #include "CommissionEmployee.h"
5  #include "BasePlusCommissionEmployee.h"
6
7  int main()
8  {
9     CommissionEmployee *commissionEmployeePtr = 0; // base class
10    BasePlusCommissionEmployee basePlusCommissionEmployee(
11       "Bob", "Lewis", "333-33-3333", 5000, .04, 300 ); // derived class
12
13    // aim base-class pointer at derived-class object
14    commissionEmployeePtr = &basePlusCommissionEmployee;
15
16    // invoke base-class member functions on derived-class
17    // object through base-class pointer (allowed)
18    string firstName = commissionEmployeePtr->getFirstName();
19    string lastName = commissionEmployeePtr->getLastName();
20    string ssn = commissionEmployeePtr->getSocialSecurityNumber();
21    double grossSales = commissionEmployeePtr->getGrossSales();
22    double commissionRate = commissionEmployeePtr->getCommissionRate();
```

Fig. 13.7 | Attempting to invoke derived-class-only functions via a base-class pointer. (Part 1 of 3.)

```
23
24      // attempt to invoke derived-class-only member functions
25      // on derived-class object through base-class pointer (disallowed)
26      double baseSalary = commissionEmployeePtr->getBaseSalary();
27      commissionEmployeePtr->setBaseSalary( 500 );
28  } // end main
```

Fig. 13.7 | Attempting to invoke derived-class-only functions via a base-class pointer. (Part 2 of 3.)

Review summary 1

- BaseObject
  - Call Based function OK
  - Call Derived function ERROR
- DerivedObject
  - Call Derived function ---OK
  - Call Base function ----OK
- BasePtr to BaseObject
  - Same as BaseObject
- DerivedPtr to DerivedObject
  - Same as DerivedObject

Review summary 2

- BasePtr to DerivedObject
  - Call Based function OK
  - Call Derived-only function ERROR
- DerivedPtr to BaseObject
  - Syntax ERROR

13.3.4:Virtual Function

Virtual function

- Which version of virtual function to call
  - Determine by
    - The type of object being pointed to
    - NOT by the type of the handle
- Example: call virtual function
  - static binding
    - base-class object
    - derived-class object
  - dynamic binding
    - base-class pointer to base-class object
    - derived-class pointer to derived-class object
    - base-class pointer to derived-class object

```
1  // Fig. 13.8: CommissionEmployee.h
2  // CommissionEmployee class definition represents a commission employee.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // C++ standard string class
7  using namespace std;
8
9  class CommissionEmployee
10 {
11 public:
12    CommissionEmployee( const string &, const string &, const string &,
13       double = 0.0, double = 0.0 );
14
15    void setFirstName( const string & ); // set first name
16    string getFirstName() const; // return first name
17
18    void setLastName( const string & ); // set last name
19    string getLastName() const; // return last name
20
21    void setSocialSecurityNumber( const string & ); // set SSN
22    string getSocialSecurityNumber() const; // return SSN
```

Fig. 13.8 | `CommissionEmployee` class header file declares `earnings` and `print` functions as `virtual`. (Part 1 of 2.)

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

Fig. 13.8 | `CommissionEmployee` class header file declares `earnings` and `print` functions as `virtual`. (Part 2 of 2.)

```
1  // Fig. 13.9: 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  using namespace std;
10
11 class BasePlusCommissionEmployee : public CommissionEmployee
12 {
13 public:
14    BasePlusCommissionEmployee( const string &, const string &,
15       const string &, double = 0.0, double = 0.0, double = 0.0 );
16
17    void setBaseSalary( double ); // set base salary
18    double getBaseSalary() const; // return base salary
19
20    virtual double earnings() const; // calculate earnings
21    virtual void print() const; // print BasePlusCommissionEmployee object
```

Fig. 13.9 | `BasePlusCommissionEmployee` class header file declares `earnings` and `print` functions as `virtual`. (Part 1 of 2.)

```
22   private:
23      double baseSalary; // base salary
24   }; // end class BasePlusCommissionEmployee
25
26   #endif
```

Fig. 13.9 | `BasePlusCommissionEmployee` class header file declares `earnings` and `print` functions as `virtual`. (Part 2 of 2.)

```
1  // Fig. 13.10: fig13_10.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 = 0;
17
18    // create derived-class object
19    BasePlusCommissionEmployee basePlusCommissionEmployee(
20       "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
21
```

Fig. 13.10 | 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 = 0;
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";
```

Fig. 13.10 | Demonstrating polymorphism by invoking a derived-class `virtual` function via a base-class pointer to a derived-class object. (Part 2 of 5.)

```
44      commissionEmployeePtr->print(); // invokes base-class print
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. 13.10 | Demonstrating polymorphism by invoking a derived-class `virtual` function via a base-class pointer to a derived-class object. (Part 3 of 5.)