Object-Oriented Programming: Inheritance

OBJECTIVES

In this chapter you will learn:

- To create classes by inheriting from existing classes.
- How inheritance promotes software reuse.
- The notions of base classes and derived classes and the relationships between them.
- The `protected` member access specifier.
- The use of constructors and destructors in inheritance hierarchies.
- The differences between `public`, `protected` and `private` inheritance.
- The use of inheritance to customize existing software.

Self-Review Exercises

23.1   Fill in the blanks in each of the following statements:

a)   _____ is a form of software reuse in which new classes absorb the data and behaviors of existing classes and embellish these classes with new capabilities.

ANS: Inheritance.

b)   A base class's _____ members can be accessed only in the base-class definition or in derived-class definitions.

ANS: `protected`.

c)   In a(n) _____ relationship, an object of a derived class also can be treated as an object of its base class.

ANS: *is-a* or inheritance.

d)   In a(n) _____ relationship, a class object has one or more objects of other classes as members.

ANS: *has-a* or composition or aggregation.

e)   In single inheritance, a class exists in a(n) _____ relationship with its derived classes.

ANS: hierarchical.

f)   A base class's _____ members are accessible within that base class and anywhere that the program has a handle to an object of that base class or to an object of one of its derived classes.

ANS: `public`.

g)   A base class's `protected` access members have a level of protection between those of `public` and _____ access.

ANS: `private`.

h)   C++ provides for _____, which allows a derived class to inherit from many base classes, even if these base classes are unrelated.

ANS: multiple inheritance.

i)   When an object of a derived class is instantiated, the base class's _____ is called implicitly or explicitly to do any necessary initialization of the base-class data members in the derived-class object.

ANS: constructor.

j)   When deriving a class from a base class with `public` inheritance, `public` members of the base class become _____ members of the derived class, and `protected` members of the base class become _____ members of the derived class.

ANS: `public`, `protected`.

k)   When deriving a class from a base class with `protected` inheritance, `public` members of the base class become _____ members of the derived class, and `protected` members of the base class become _____ members of the derived class.

ANS: `protected`, `protected`.

23.2   State whether each of the following is *true* or *false*. If *false*, explain why.

a)   Base-class constructors are not inherited by derived classes.

ANS: True.

b)   A *has-a* relationship is implemented via inheritance.

ANS: False. A *has-a* relationship is implemented via composition. An *is-a* relationship is implemented via inheritance.

c)   A `Car` class has an *is-a* relationship with the `SteeringWheel` and `Brakes` classes.

ANS: False. This is an example of a *has-a* relationship. Class `Car` has an *is-a* relationship with class `Vehicle`.

d)   Inheritance encourages the reuse of proven high-quality software.

ANS: True.

e) When a derived-class object is destroyed, the destructors are called in the reverse order of the constructors.
**ANS:** True.

Exercises

23.3 Discuss the ways in which inheritance promotes software reuse, saves time during program development and helps prevent errors.

> **ANS:** Inheritance allows developers to create derived classes that reuse code declared already in a base class. Avoiding the duplication of common functionality between several classes by building an inheritance hierarchy to contain the classes can save developers a considerable amount of time. Similarly, placing common functionality in a single base class, rather than duplicating the code in multiple unrelated classes, helps prevent the same errors from appearing in multiple source-code files. If several classes each contain duplicate code containing an error, the software developer has to spend time correcting each source-code file with the error. However, if these classes take advantage of inheritance, and the error occurs in the common functionality of the base class, the software developer needs to modify only the base class's code.

23.4 Draw an inheritance hierarchy for students at a university similar to the hierarchy shown in Fig. 23.2. Use Student as the base class of the hierarchy, then include classes UndergraduateStudent and GraduateStudent that derive from Student. Continue to extend the hierarchy as deep (i.e., as many levels) as possible. For example, Freshman, Sophomore, Junior and Senior might derive from UndergraduateStudent, and DoctoralStudent and MastersStudent might derive from GraduateStudent. After drawing the hierarchy, discuss the relationships that exist between the classes. [*Note:* You do not need to write any code for this exercise.]

> **ANS:**
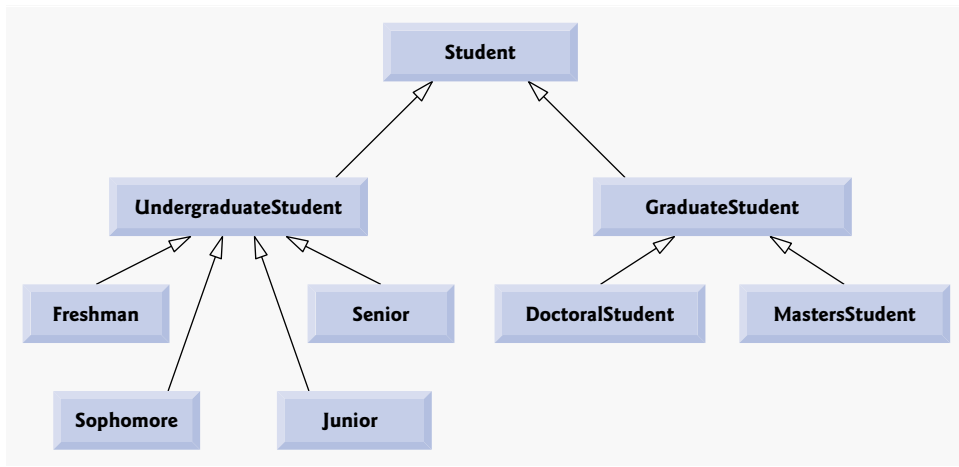
This hierarchy contains many "is-a" (inheritance) relationships. An UndergraduateStudent *is a* Student. A GraduateStudent *is a* Student too. Each of the classes Freshman, Sophomore, Junior and Senior *is an* UndergraduateStudent and *is a* Student. Each of the classes DoctoralStudent and MastersStudent *is a* GraduateStudent and *is a* Student.

23.5 (*Package Inheritance Hierarchy*) Package-delivery services, such as FedEx®, DHL® and UPS®, offer a number of different shipping options, each with specific costs associated. Create an inheritance hierarchy to represent various types of packages. Use `Package` as the base class of the hierarchy, then include classes `TwoDayPackage` and `OvernightPackage` that derive from `Package`. Base class `Package` should include data members representing the name, address, city, state and ZIP code for both the sender and the recipient of the package, in addition to data members that store the weight (in ounces) and cost per ounce to ship the package. `Package`'s constructor should initialize these data members. Ensure that the weight and cost per ounce contain positive values. `Package` should provide a `public` member function `calculateCost` that returns a `double` indicating the cost associated with shipping the package. `Package`'s `calculateCost` function should determine the cost by multiplying the weight by the cost per ounce. Derived class `TwoDayPackage` should inherit the functionality of base class `Package`, but also include a data member that represents a flat fee that the shipping company charges for two-day-delivery service. `TwoDayPackage`'s constructor should receive a value to initialize this data member. `TwoDayPackage` should redefine member function `calculateCost` so that it computes the shipping cost by adding the flat fee to the weight-based cost calculated by base class `Package`'s `calculateCost` function. Class `OvernightPackage` should inherit directly from class `Package` and contain an additional data member representing an additional fee per ounce charged for overnight-delivery service. `OvernightPackage` should redefine member function `calculateCost` so that it adds the additional fee per ounce to the standard cost per ounce before calculating the shipping cost. Write a test program that creates objects of each type of `Package` and tests member function `calculateCost`.

     ANS:

```
1   // Exercise 23.5 Solution: Package.h
2   // Definition of base class Package.
3   #ifndef PACKAGE_H
4   #define PACKAGE_H
5
6   #include <string>
7   using std::string;
8
9   class Package
10  {
11  public:
12     // constructor initiliazes data members
13     Package( const string &, const string &, const string &,
14        const string &, int, const string &, const string &, const string &,
15        const string &, int, double, double );
16
17     void setSenderName( const string & ); // set sender's name
18     string getSenderName() const; // return sender's name
19     void setSenderAddress( const string & ); // set sender's address
20     string getSenderAddress() const; // return sender's address
21     void setSenderCity( const string & ); // set sender's city
22     string getSenderCity() const; // return sender's city
23     void setSenderState( const string & ); // set sender's state
24     string getSenderState() const; // return sender's state
25     void setSenderZIP( int ); // set sender's ZIP code
26     int getSenderZIP() const; // return sender's ZIP code
27     void setRecipientName( const string & ); // set recipient's name
28     string getRecipientName() const; // return recipient's name
29     void setRecipientAddress( const string & ); // set recipient's address
30     string getRecipientAddress() const; // return recipient's address
```

```
31       void setRecipientCity( const string & ); // set recipient's city
32       string getRecipientCity() const; // return recipient's city
33       void setRecipientState( const string & ); // set recipient's state
34       string getRecipientState() const; // return recipient's state
35       void setRecipientZIP( int ); // set recipient's ZIP code
36       int getRecipientZIP() const; // return recipient's ZIP code
37       void setWeight( double ); // validate and store weight
38       double getWeight() const; // return weight of package
39       void setCostPerOunce( double ); // validate and store cost per ounce
40       double getCostPerOunce() const; // return cost per ounce
41
42       double calculateCost() const; // calculate shipping cost for package
43    private:
44       // data members to store sender and recipient's address information
45       string senderName;
46       string senderAddress;
47       string senderCity;
48       string senderState;
49       int senderZIP;
50       string recipientName;
51       string recipientAddress;
52       string recipientCity;
53       string recipientState;
54       int recipientZIP;
55
56       double weight; // weight of the package
57       double costPerOunce; // cost per ounce to ship the package
58    }; // end class Package
59
60    #endif
```

```
 1    // Exercise 23.5 Solution: Package.cpp
 2    // Member-function definitions of class Package.
 3
 4    #include "Package.h" // Package class definition
 5
 6    // constructor initiliazes data members
 7    Package::Package( const string &sName, const string &sAddress,
 8       const string &sCity, const string &sState, int sZIP,
 9       const string &rName, const string &rAddress, const string &rCity,
10       const string &rState, int rZIP, double w, double cost )
11       : senderName( sName ), senderAddress( sAddress ), senderCity( sCity ),
12         senderState( sState ), senderZIP( sZIP ), recipientName( rName ),
13         recipientAddress( rAddress ), recipientCity( rCity ),
14         recipientState( rState ), recipientZIP( rZIP )
15    {
16       setWeight( w ); // validate and store weight
17       setCostPerOunce( cost ); // validate and store cost per ounce
18    } // end Package constructor
19
20    // set sender's name
21    void Package::setSenderName( const string &name )
```

```
22   {
23       senderName = name;
24   } // end function setSenderName
25
26   // return sender's name
27   string Package::getSenderName() const
28   {
29       return senderName;
30   } // end function getSenderName
31
32   // set sender's address
33   void Package::setSenderAddress( const string &address )
34   {
35       senderAddress = address;
36   } // end function setSenderAddress
37
38   // return sender's address
39   string Package::getSenderAddress() const
40   {
41       return senderAddress;
42   } // end function getSenderAddress
43
44   // set sender's city
45   void Package::setSenderCity( const string &city )
46   {
47       senderCity = city;
48   } // end function setSenderCity
49
50   // return sender's city
51   string Package::getSenderCity() const
52   {
53       return senderCity;
54   } // end function getSenderCity
55
56   // set sender's state
57   void Package::setSenderState( const string &state )
58   {
59       senderState = state;
60   } // end function setSenderState
61
62   // return sender's state
63   string Package::getSenderState() const
64   {
65       return senderState;
66   } // end function getSenderState
67
68   // set sender's ZIP code
69   void Package::setSenderZIP( int zip )
70   {
71       senderZIP = zip;
72   } // end function setSenderZIP
73
```

```
74   // return sender's ZIP code
75   int Package::getSenderZIP() const
76   {
77      return senderZIP;
78   } // end function getSenderZIP
79
80   // set recipient's name
81   void Package::setRecipientName( const string &name )
82   {
83       recipientName = name;
84   } // end function setRecipientName
85
86   // return recipient's name
87   string Package::getRecipientName() const
88   {
89      return recipientName;
90   } // end function getRecipientName
91
92   // set recipient's address
93   void Package::setRecipientAddress( const string &address )
94   {
95      recipientAddress = address;
96   } // end function setRecipientAddress
97
98   // return recipient's address
99   string Package::getRecipientAddress() const
100  {
101     return recipientAddress;
102  } // end function getRecipientAddress
103
104  // set recipient's city
105  void Package::setRecipientCity( const string &city )
106  {
107     recipientCity = city;
108  } // end function setRecipientCity
109
110  // return recipient's city
111  string Package::getRecipientCity() const
112  {
113     return recipientCity;
114  } // end function getRecipientCity
115
116  // set recipient's state
117  void Package::setRecipientState( const string &state )
118  {
119     recipientState = state;
120  } // end function setRecipientState
121
122  // return recipient's state
123  string Package::getRecipientState() const
124  {
125     return recipientState;
126  } // end function getRecipientState
127
```

```
128  // set recipient's ZIP code
129  void Package::setRecipientZIP( int zip )
130  {
131     recipientZIP = zip;
132  } // end function setRecipientZIP
133
134  // return recipient's ZIP code
135  int Package::getRecipientZIP() const
136  {
137     return recipientZIP;
138  } // end function getRecipientZIP
139
140  // validate and store weight
141  void Package::setWeight( double w )
142  {
143     weight = ( w < 0.0 ) ? 0.0 : w;
144  } // end function setWeight
145
146  // return weight of package
147  double Package::getWeight() const
148  {
149     return weight;
150  } // end function getWeight
151
152  // validate and store cost per ounce
153  void Package::setCostPerOunce( double cost )
154  {
155     costPerOunce = ( cost < 0.0 ) ? 0.0 : cost;
156  } // end function setCostPerOunce
157
158  // return cost per ounce
159  double Package::getCostPerOunce() const
160  {
161     return costPerOunce;
162  } // end function getCostPerOunce
163
164  // calculate shipping cost for package
165  double Package::calculateCost() const
166  {
167     return getWeight() * getCostPerOunce();
168  } // end function calculateCost
```

```
1   // Exercise 23.5 Solution: TwoDayPackage.h
2   // Definition of derived class TwoDayPackage.
3   #ifndef TWODAY_H
4   #define TWODAY_H
5
6   #include "Package.h" // Package class definition
7
8   class TwoDayPackage : public Package
9   {
10  public:
11     TwoDayPackage( const string &, const string &, const string &,
12        const string &, int, const string &, const string &, const string &,
13        const string &, int, double, double, double );
```

```
14
15      void setFlatFee( double ); // set flat fee for two-day-delivery service
16      double getFlatFee() const; // return flat fee
17
18      double calculateCost() const; // calculate shipping cost for package
19   private:
20      double flatFee; // flat fee for two-day-delivery service
21   }; // end class TwoDayPackage
22
23   #endif
```

```cpp
1    // Exercise 23.5 Solution: TwoDayPackage.cpp
2    // Member-function definitions of class TwoDayPackage.
3
4    #include "TwoDayPackage.h" // TwoDayPackage class definition
5
6    // constructor
7    TwoDayPackage::TwoDayPackage( const string &sName,
8       const string &sAddress, const string &sCity, const string &sState,
9       int sZIP, const string &rName, const string &rAddress,
10      const string &rCity, const string &rState, int rZIP,
11      double w, double cost, double fee )
12      : Package( sName, sAddress, sCity, sState, sZIP,
13         rName, rAddress, rCity, rState, rZIP, w, cost )
14   {
15      setFlatFee( fee );
16   } // end TwoDayPackage constructor
17
18   // set flat fee
19   void TwoDayPackage::setFlatFee( double fee )
20   {
21      flatFee = ( fee < 0.0 ) ? 0.0 : fee;
22   } // end function setFlatFee
23
24   // return flat fee
25   double TwoDayPackage::getFlatFee() const
26   {
27      return flatFee;
28   } // end function getFlatFee
29
30   // calculate shipping cost for package
31   double TwoDayPackage::calculateCost() const
32   {
33      return Package::calculateCost() + getFlatFee();
34   } // end function calculateCost
```

```cpp
1    // Exercise 23.5 Solution: OvernightPackage.h
2    // Definition of derived class OvernightPackage.
3    #ifndef OVERNIGHT_H
4    #define OVERNIGHT_H
5
6    #include "Package.h" // Package class definition
7
```

```
 8   class OvernightPackage : public Package
 9   {
10   public:
11      OvernightPackage( const string &, const string &, const string &,
12         const string &, int, const string &, const string &, const string &,
13         const string &, int, double, double, double );
14
15      void setOvernightFeePerOunce( double ); // set overnight fee
16      double getOvernightFeePerOunce() const; // return overnight fee
17
18      double calculateCost() const; // calculate shipping cost for package
19   private:
20      double overnightFeePerOunce; // fee per ounce for overnight delivery
21   }; // end class OvernightPackage
22
23   #endif
```

```
 1   // Exercise 23.5 Solution: OvernightPackage.cpp
 2   // Member-function definitions of class OvernightPackage.
 3
 4   #include "OvernightPackage.h" // OvernightPackage class definition
 5
 6   // constructor
 7   OvernightPackage::OvernightPackage( const string &sName,
 8      const string &sAddress, const string &sCity, const string &sState,
 9      int sZIP, const string &rName, const string &rAddress,
10      const string &rCity, const string &rState, int rZIP,
11      double w, double cost, double fee )
12      : Package( sName, sAddress, sCity, sState, sZIP,
13         rName, rAddress, rCity, rState, rZIP, w, cost )
14   {
15      setOvernightFeePerOunce( fee ); // validate and store overnight fee
16   } // end OvernightPackage constructor
17
18   // set overnight fee
19   void OvernightPackage::setOvernightFeePerOunce( double overnightFee )
20   {
21      overnightFeePerOunce = ( overnightFee < 0.0 ) ? 0.0 : overnightFee;
22   } // end function setOvernightFeePerOunce
23
24   // return overnight fee
25   double OvernightPackage::getOvernightFeePerOunce() const
26   {
27      return overnightFeePerOunce;
28   } // end function getOvernghtFeePerOunce
29
30   // calculate shipping cost for package
31   double OvernightPackage::calculateCost() const
32   {
33      return getWeight() * ( getCostPerOunce() + getOvernightFeePerOunce() );
34   } // end function calculateCost
```

```cpp
1   // Exercise 23.5 Solution: ex23_05.cpp
2   // Driver program for Package hierarchy.
3   #include <iostream>
4   using std::cout;
5   using std::endl;
6
7   #include <iomanip>
8   using std::setprecision;
9   using std::fixed;
10
11  #include "Package.h" // Package class definition
12  #include "TwoDayPackage.h" // TwoDayPackage class definition
13  #include "OvernightPackage.h" // OvernightPackage class definition
14
15  int main()
16  {
17     Package package1( "Lou Brown", "1 Main St", "Boston", "MA", 11111,
18        "Mary Smith", "7 Elm St", "New York", "NY", 22222, 8.5, .5 );
19     TwoDayPackage package2( "Lisa Klein", "5 Broadway", "Somerville", "MA",
20        33333, "Bob George", "21 Pine Rd", "Cambridge", "MA", 44444,
21        10.5, .65, 2.0 );
22     OvernightPackage package3( "Ed Lewis", "2 Oak St", "Boston", "MA",
23        55555, "Don Kelly", "9 Main St", "Denver", "CO", 66666,
24        12.25, .7, .25 );
25
26     cout << fixed << setprecision( 2 );
27
28     // print each package's information and cost
29     cout << "Package 1:\n\nSender:\n" << package1.getSenderName()
30        << '\n' << package1.getSenderAddress() << '\n'
31        << package1.getSenderCity() << ", " << package1.getSenderState()
32        << ' ' << package1.getSenderZIP();
33     cout << "\n\nRecipient:\n" << package1.getRecipientName()
34        << '\n' << package1.getRecipientAddress() << '\n'
35        << package1.getRecipientCity() << ", "
36        << package1.getRecipientState() << ' '
37        << package1.getRecipientZIP();
38     cout << "\n\nCost: $" << package1.calculateCost() << endl;
39
40     cout << "\nPackage 2:\n\nSender:\n" << package2.getSenderName()
41        << '\n' << package2.getSenderAddress() << '\n'
42        << package2.getSenderCity() << ", " << package2.getSenderState()
43        << ' ' << package2.getSenderZIP();
44     cout << "\n\nRecipient:\n" << package2.getRecipientName()
45        << '\n' << package2.getRecipientAddress() << '\n'
46        << package2.getRecipientCity() << ", "
47        << package2.getRecipientState() << ' '
48        << package2.getRecipientZIP();
49     cout << "\n\nCost: $" << package2.calculateCost() << endl;
50
51     cout << "\nPackage 3:\n\nSender:\n" << package3.getSenderName()
52        << '\n' << package3.getSenderAddress() << '\n'
53        << package3.getSenderCity() << ", " << package3.getSenderState()
54        << ' ' << package3.getSenderZIP();
```

```
55        cout << "\n\nRecipient:\n" << package3.getRecipientName()
56            << '\n' << package3.getRecipientAddress() << '\n'
57            << package3.getRecipientCity() << ", "
58            << package3.getRecipientState() << ' '
59            << package3.getRecipientZIP();
60        cout << "\n\nCost: $" << package3.calculateCost() << endl;
61        return 0;
62    } // end main
```

```
Package 1:

Sender:
Lou Brown
1 Main St
Boston, MA 11111

Recipient:
Mary Smith
7 Elm St
New York, NY 22222

Cost: $4.25

Package 2:

Sender:
Lisa Klein
5 Broadway
Somerville, MA 33333

Recipient:
Bob George
21 Pine Rd
Cambridge, MA 44444

Cost: $8.82

Package 3:

Sender:
Ed Lewis
2 Oak St
Boston, MA 55555

Recipient:
Don Kelly
9 Main St
Denver, CO 66666

Cost: $11.64
```

23.6   *(Account Inheritance Hierarchy)* Create an inheritance hierarchy that a bank might use to represent customers' bank accounts. All customers at this bank can deposit (i.e., credit) money into their accounts and withdraw (i.e., debit) money from their accounts. More specific types of accounts also exist. Savings accounts, for instance, earn interest on the money they hold. Checking accounts, on the other hand, charge a fee per transaction (i.e., credit or debit).

Create an inheritance hierarchy containing base class `Account` and derived classes `Savings-Account` and `CheckingAccount` that inherit from class `Account`. Base class `Account` should include one data member of type `double` to represent the account balance. The class should provide a constructor that receives an initial balance and uses it to initialize the data member. The constructor should validate the initial balance to ensure that it is greater than or equal to `0.0`. If not, the balance should be set to `0.0` and the constructor should display an error message, indicating that the initial balance was invalid. The class should provide three member functions. Member function `credit` should add an amount to the current balance. Member function `debit` should withdraw money from the `Account` and ensure that the debit amount does not exceed the `Account`'s balance. If it does, the balance should be left unchanged and the function should print the message `"Debit amount exceeded account balance."` Member function `getBalance` should return the current balance.

Derived class `SavingsAccount` should inherit the functionality of an `Account`, but also include a data member of type `double` indicating the interest rate (percentage) assigned to the `Account`. `SavingsAccount`'s constructor should receive the initial balance, as well as an initial value for the `SavingsAccount`'s interest rate. `SavingsAccount` should provide a `public` member function `calculateInterest` that returns a `double` indicating the amount of interest earned by an account. Member function `calculateInterest` should determine this amount by multiplying the interest rate by the account balance. [*Note:* `SavingsAccount` should inherit member functions `credit` and `debit` as is without redefining them.]

Derived class `CheckingAccount` should inherit from base class `Account` and include an additional data member of type `double` that represents the fee charged per transaction. `CheckingAccount`'s constructor should receive the initial balance, as well as a parameter indicating a fee amount. Class `CheckingAccount` should redefine member functions `credit` and `debit` so that they subtract the fee from the account balance whenever either transaction is performed successfully. `CheckingAccount`'s versions of these functions should invoke the base-class `Account` version to perform the updates to an account balance. `CheckingAccount`'s `debit` function should charge a fee only if money is actually withdrawn (i.e., the debit amount does not exceed the account balance). [*Hint:* Define `Account`'s `debit` function so that it returns a `bool` indicating whether money was withdrawn. Then use the return value to determine whether a fee should be charged.]

After defining the classes in this hierarchy, write a program that creates objects of each class and tests their member functions. Add interest to the `SavingsAccount` object by first invoking its `calculateInterest` function, then passing the returned interest amount to the object's `credit` function.

ANS:

```
1   // Solution 23.10 Solution: Account.h
2   // Definition of Account class.
3   #ifndef ACCOUNT_H
4   #define ACCOUNT_H
5
6   class Account
7   {
8   public:
9      Account( double ); // constructor initializes balance
10     void credit( double ); // add an amount to the account balance
11     bool debit( double ); // subtract an amount from the account balance
12     void setBalance( double ); // sets the account balance
13     double getBalance(); // return the account balance
14  private:
15     double balance; // data member that stores the balance
16  }; // end class Account
17
18  #endif
```

```cpp
1   // Exercise 23.6 Solution: Account.cpp
2   // Member-function definitions for class Account.
3   #include <iostream>
4   using std::cout;
5   using std::endl;
6
7   #include "Account.h" // include definition of class Account
8
9   // Account constructor initializes data member balance
10  Account::Account( double initialBalance )
11  {
12     // if initialBalance is greater than or equal to 0.0, set this value
13     // as the balance of the Account
14     if ( initialBalance >= 0.0 )
15        balance = initialBalance;
16     else // otherwise, output message and set balance to 0.0
17     {
18        cout << "Error: Initial balance cannot be negative." << endl;
19        balance = 0.0;
20     } // end if...else
21  } // end Account constructor
22
23  // credit (add) an amount to the account balance
24  void Account::credit( double amount )
25  {
26     balance = balance + amount; // add amount to balance
27  } // end function credit
28
29  // debit (subtract) an amount from the account balance
30  // return bool indicating whether money was debited
31  bool Account::debit( double amount )
32  {
33     if ( amount > balance ) // debit amount exceeds balance
34     {
35        cout << "Debit amount exceeded account balance." << endl;
36        return false;
37     } // end if
38     else // debit amount does not exceed balance
39     {
40        balance = balance - amount;
41        return true;
42     } // end else
43  } // end function debit
44
45  // set the account balance
46  void Account::setBalance( double newBalance )
47  {
48     balance = newBalance;
49  } // end function setBalance
50
51  // return the account balance
52  double Account::getBalance()
53  {
54     return balance;
55  } // end function getBalance
```

```
 1   // Exercise 23.6 Solution: SavingsAccount.h
 2   // Definition of SavingsAccount class.
 3   #ifndef SAVINGS_H
 4   #define SAVINGS_H
 5
 6   #include "Account.h" // Account class definition
 7
 8   class SavingsAccount : public Account
 9   {
10   public:
11      // constructor initializes balance and interest rate
12      SavingsAccount( double, double );
13
14      double calculateInterest(); // determine interest owed
15   private:
16      double interestRate; // interest rate (percentage) earned by account
17   }; // end class SavingsAccount
18
19   #endif
```

```
 1   // Exercise 23.6 Solution: SavingsAccount.cpp
 2   // Member-function definitions for class SavingsAccount.
 3
 4   #include "SavingsAccount.h" // SavingsAccount class definition
 5
 6   // constructor initializes balance and interest rate
 7   SavingsAccount::SavingsAccount( double initialBalance, double rate )
 8      : Account( initialBalance ) // initialize base class
 9   {
10      interestRate = ( rate < 0.0 ) ? 0.0 : rate; // set interestRate
11   } // end SavingsAccount constructor
12
13   // return the amount of interest earned
14   double SavingsAccount::calculateInterest()
15   {
16      return getBalance() * interestRate;
17   } // end function calculateInterest
```

```
 1   // Exercise 23.6 Solution: CheckingAccount.h
 2   // Definition of CheckingAccount class.
 3   #ifndef CHECKING_H
 4   #define CHECKING_H
 5
 6   #include "Account.h" // Account class definition
 7
 8   class CheckingAccount : public Account
 9   {
10   public:
11      // constructor initializes balance and transaction fee
12      CheckingAccount( double, double );
13
```

```
14     void credit( double ); // redefined credit function
15     bool debit( double ); // redefined debit function
16  private:
17     double transactionFee; // fee charged per transaction
18
19     // utility function to charge fee
20     void chargeFee();
21  }; // end class CheckingAccount
22
23  #endif
```

```
1   // Exercise 23.6 Solution: CheckingAccount.cpp
2   // Member-function definitions for class CheckingAccount.
3   #include <iostream>
4   using std::cout;
5   using std::endl;
6
7   #include "CheckingAccount.h" // CheckingAccount class definition
8
9   // constructor initializes balance and transaction fee
10  CheckingAccount::CheckingAccount( double initialBalance, double fee )
11     : Account( initialBalance ) // initialize base class
12  {
13     transactionFee = ( fee < 0.0 ) ? 0.0 : fee; // set transaction fee
14  } // end CheckingAccount constructor
15
16  // credit (add) an amount to the account balance and charge fee
17  void CheckingAccount::credit( double amount )
18  {
19     Account::credit( amount ); // always succeeds
20     chargeFee();
21  } // end function credit
22
23  // debit (subtract) an amount from the account balance and charge fee
24  bool CheckingAccount::debit( double amount )
25  {
26     bool success = Account::debit( amount ); // attempt to debit
27
28     if ( success ) // if money was debited, charge fee and return true
29     {
30        chargeFee();
31        return true;
32     } // end if
33     else // otherwise, do not charge fee and return false
34        return false;
35  } // end function debit
36
37  // subtract transaction fee
38  void CheckingAccount::chargeFee()
39  {
40     Account::setBalance( getBalance() - transactionFee );
41     cout << "$" << transactionFee << " transaction fee charged." << endl;
42  } // end function chargeFee
```

```cpp
1   // Exercise 23.6 Solution: ex23_6.cpp
2   // Test program for Account hierarchy.
3   #include <iostream>
4   using std::cout;
5   using std::endl;
6
7   #include <iomanip>
8   using std::setprecision;
9   using std::fixed;
10
11  #include "Account.h" // Account class definition
12  #include "SavingsAccount.h" // SavingsAccount class definition
13  #include "CheckingAccount.h" // CheckingAccount class definition
14
15  int main()
16  {
17     Account account1( 50.0 ); // create Account object
18     SavingsAccount account2( 25.0, .03 ); // create SavingsAccount object
19     CheckingAccount account3( 80.0, 1.0 ); // create CheckingAccount object
20
21     cout << fixed << setprecision( 2 );
22
23     // display initial balance of each object
24     cout << "account1 balance: $" << account1.getBalance() << endl;
25     cout << "account2 balance: $" << account2.getBalance() << endl;
26     cout << "account3 balance: $" << account3.getBalance() << endl;
27
28     cout << "\nAttempting to debit $25.00 from account1." << endl;
29     account1.debit( 25.0 ); // try to debit $25.00 from account1
30     cout << "\nAttempting to debit $30.00 from account2." << endl;
31     account2.debit( 30.0 ); // try to debit $30.00 from account2
32     cout << "\nAttempting to debit $40.00 from account3." << endl;
33     account3.debit( 40.0 ); // try to debit $40.00 from account3
34
35     // display balances
36     cout << "\naccount1 balance: $" << account1.getBalance() << endl;
37     cout << "account2 balance: $" << account2.getBalance() << endl;
38     cout << "account3 balance: $" << account3.getBalance() << endl;
39
40     cout << "\nCrediting $40.00 to account1." << endl;
41     account1.credit( 40.0 ); // credit $40.00 to account1
42     cout << "\nCrediting $65.00 to account2." << endl;
43     account2.credit( 65.0 ); // credit $65.00 to account2
44     cout << "\nCrediting $20.00 to account3." << endl;
45     account3.credit( 20.0 ); // credit $20.00 to account3
46
47     // display balances
48     cout << "\naccount1 balance: $" << account1.getBalance() << endl;
49     cout << "account2 balance: $" << account2.getBalance() << endl;
50     cout << "account3 balance: $" << account3.getBalance() << endl;
51
52     // add interest to SavingsAccount object account2
53     double interestEarned = account2.calculateInterest();
54     cout << "\nAdding $" << interestEarned << " interest to account2."
```

```
55          << endl;
56       account2.credit( interestEarned );
57
58       cout << "\nNew account2 balance: $" << account2.getBalance() << endl;
59       return 0;
60    } // end main
```

```
account1 balance: $50.00
account2 balance: $25.00
account3 balance: $80.00

Attempting to debit $25.00 from account1.

Attempting to debit $30.00 from account2.
Debit amount exceeded account balance.

Attempting to debit $40.00 from account3.
$1.00 transaction fee charged.

account1 balance: $25.00
account2 balance: $25.00
account3 balance: $39.00

Crediting $40.00 to account1.

Crediting $65.00 to account2.

Crediting $20.00 to account3.
$1.00 transaction fee charged.

account1 balance: $65.00
account2 balance: $90.00
account3 balance: $58.00

Adding $2.70 interest to account2.

New account2 balance: $92.70
```