9.10 `const` Objects and `const` Member Functions (cont.)

- A constructor *must* be allowed to modify an object so that the object can be initialized properly.
- A destructor must be able to perform its termination housekeeping chores before an object's memory is reclaimed by the system.
- Attempting to declare a constructor or destructor `const` is a compilation error.
- The "`const`ness" of a `const` object is enforced from the time the constructor *completes* initialization of the object until that object's destructor is called.

9.10 `const` Objects and `const` Member Functions (cont.)

Using `const` and Non-`const` Member Functions

- The program of Fig. 9.16 uses class `Time` from Figs. 9.4–9.5, but removes `const` from function `printStandard`'s prototype and definition so that we can show a compilation error.

```
1  // Fig. 9.16: fig09_16.cpp
2  // const objects and const member functions.
3  #include "Time.h" // include Time class definition
4
5  int main()
6  {
7     Time wakeUp( 6, 45, 0 ); // non-constant object
8     const Time noon( 12, 0, 0 ); // constant object
9
10                              // OBJECT      MEMBER FUNCTION
11    wakeUp.setHour( 18 );     // non-const   non-const
12
13    noon.setHour( 12 );       // const       non-const
14
15    wakeUp.getHour();         // non-const   const
16
17    noon.getMinute();         // const       const
18    noon.printUniversal();    // const       const
19
20    noon.printStandard();     // const       non-const
21 } // end main
```

Fig. 9.16 | `const` objects and `const` member functions. (Part 1 of 2.)

Microsoft Visual C++ compiler error messages:

```
C:\examples\ch09\Fig09_16_18\fig09_18.cpp(13) : error C2662:
   'Time::setHour' : cannot convert 'this' pointer from 'const Time' to
   'Time &'
         Conversion loses qualifiers
C:\examples\ch09\Fig09_16_18\fig09_18.cpp(20) : error C2662:
   'Time::printStandard' : cannot convert 'this' pointer from 'const Time' to
   'Time &'
       Conversion loses qualifiers
```

Fig. 9.16 | `const` objects and `const` member functions. (Part 2 of 2.)

9.11 Composition: Objects as Members of Classes

- An `AlarmClock` object needs to know when it's supposed to sound its alarm, so why not include a `Time` object as a member of the `AlarmClock` class?
- Such a capability is called composition and is sometimes referred to as a *has-a* relationship—*a class can have objects of other classes as members*.
- The next program uses classes `Date` (Figs. 9.17–9.18) and `Employee` (Figs. 9.19–9.20) to demonstrate composition.

Software Engineering Observation 9.9

A common form of software reusability is composition, in which a class has objects of other types as members.

Software Engineering Observation 9.10

Data members are constructed in the order in which they're declared in the class definition (not in the order they're listed in the constructor's member initializer list) and before their enclosing class objects (sometimes called host objects) are constructed.

```
1  // Fig. 9.17: Date.h
2  // Date class definition; Member functions defined in Date.cpp
3  #ifndef DATE_H
4  #define DATE_H
5
6  class Date
7  {
8  public:
9     static const unsigned int monthsPerYear = 12; // months in a year
10    explicit Date( int = 1, int = 1, int = 1900 ); // default constructor
11    void print() const; // print date in month/day/year format
12    ~Date(); // provided to confirm destruction order
13 private:
14    unsigned int month; // 1-12 (January-December)
15    unsigned int day; // 1-31 based on month
16    unsigned int year; // any year
17
18    // utility function to check if day is proper for month and year
19    unsigned int checkDay( int ) const;
20 }; // end class Date
21
22 #endif
```

Fig. 9.17 | Date class definition.

```
1  // Fig. 9.18: Date.cpp
2  // Date class member-function definitions.
3  #include <array>
4  #include <iostream>
5  #include <stdexcept>
6  #include "Date.h" // include Date class definition
7  using namespace std;
8
9  // constructor confirms proper value for month; calls
10 // utility function checkDay to confirm proper value for day
11 Date::Date( int mn, int dy, int yr )
12 {
13    if ( mn > 0 && mn <= monthsPerYear ) // validate the month
14       month = mn;
15    else
16       throw invalid_argument( "month must be 1-12" );
17
18    year = yr; // could validate yr
19    day = checkDay( dy ); // validate the day
20
21    // output Date object to show when its constructor is called
22    cout << "Date object constructor for date ";
23    print();
24    cout << endl;
25 } // end Date constructor
```

Fig. 9.18 | Date class member-function definitions. (Part 1 of 3.)

```
26
27  // print Date object in form month/day/year
28  void Date::print() const
29  {
30     cout << month << '/' << day << '/' << year;
31  } // end function print
32
33  // output Date object to show when its destructor is called
34  Date::~Date()
35  {
36     cout << "Date object destructor for date ";
37     print();
38     cout << endl;
39  } // end ~Date destructor
40
```

Fig. 9.18 | `Date` class member-function definitions. (Part 2 of 3.)

```
41  // utility function to confirm proper day value based on
42  // month and year; handles leap years, too
43  unsigned int Date::checkDay( int testDay ) const
44  {
45     static const array< int, monthsPerYear + 1 > daysPerMonth =
46        { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
47
48     // determine whether testDay is valid for specified month
49     if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
50        return testDay;
51
52     // February 29 check for leap year
53     if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
54        ( year % 4 == 0 && year % 100 != 0 ) ) )
55        return testDay;
56
57     throw invalid_argument( "Invalid day for current month and year" );
58  } // end function checkDay
```

Fig. 9.18 | `Date` class member-function definitions. (Part 3 of 3.)

```
1  // Fig. 9.19: Employee.h
2  // Employee class definition showing composition.
3  // Member functions defined in Employee.cpp.
4  #ifndef EMPLOYEE_H
5  #define EMPLOYEE_H
6
7  #include <string>
8  #include "Date.h" // include Date class definition
9
10 class Employee
11 {
12 public:
13    Employee( const std::string &, const std::string &,
14       const Date &, const Date & );
15    void print() const;
16    ~Employee(); // provided to confirm destruction order
17 private:
18    std::string firstName; // composition: member object
19    std::string lastName; // composition: member object
20    const Date birthDate; // composition: member object
21    const Date hireDate; // composition: member object
22 }; // end class Employee
23
24 #endif
```

Fig. 9.19 | Employee class definition showing composition.

```
1  // Fig. 9.20: Employee.cpp
2  // Employee class member-function definitions.
3  #include <iostream>
4  #include "Employee.h" // Employee class definition
5  #include "Date.h" // Date class definition
6  using namespace std;
7
8  // constructor uses member initializer list to pass initializer
9  // values to constructors of member objects
10 Employee::Employee( const string &first, const string &last,
11    const Date &dateOfBirth, const Date &dateOfHire )
12    : firstName( first ), // initialize firstName
13      lastName( last ), // initialize lastName
14      birthDate( dateOfBirth ), // initialize birthDate
15      hireDate( dateOfHire ) // initialize hireDate
16 {
17    // output Employee object to show when constructor is called
18    cout << "Employee object constructor: "
19       << firstName << ' ' << lastName << endl;
20 } // end Employee constructor
21
```

Fig. 9.20 | Employee class member-function definitions. (Part 1 of 2.)

```
22  // print Employee object
23  void Employee::print() const
24  {
25     cout << lastName << ", " << firstName << "  Hired: ";
26     hireDate.print();
27     cout << "  Birthday: ";
28     birthDate.print();
29     cout << endl;
30  } // end function print
31
32  // output Employee object to show when its destructor is called
33  Employee::~Employee()
34  {
35     cout << "Employee object destructor: "
36        << lastName << ", " << firstName << endl;
37  } // end ~Employee destructor
```

Fig. 9.20 | `Employee` class member-function definitions. (Part 2 of 2.)

9.11 Composition: Objects as Members of Classes (cont.)

`Employee` Constructor's Member Initializer List

- The *colon (`:`)* following the constructor's header (Fig. 9.20, line 12) begins the *member initializer list*.
- The member initializers specify the `Employee` constructor parameters being passed to the constructors of the `string` and `Date` data members.
- Again, member initializers are separated by commas.
- The order of the member initializers does not matter.
- They're executed in the order that the member objects are declared in class Employee.