Complementos de Programação de Computadores – Aula 2
Classes e Abstracção de Dados

Mestrado Integrado em Electrónica Industrial e Computadores

Luís Paulo Reis

lpreis1970@gmail.com / lpreis@dsi.uminho.pt

Professor Associado do Departamento de Sistemas de Informação, Escola de Engenharia, Universidade do Minho, Portugal

(Slides Baseados em P.Deitel e H.Deitel 2010 e Paulo Cortez, 2011)

Outline

- 16.1 Introduction
- 16.2 Implementing a Time Abstract Data Type with a Class
- 16.3 Class Scope and Accessing Class Members
- 16.4 Separating Interface from Implementation
- 16.5 Controlling Access to Members
- 16.6 Access Functions and Utility Functions
- 16.7 Initializing Class Objects: Constructors
- 16.8 Using Default Arguments with Constructors
- 16.9 Using Destructors
- 16.10 Calling Constructors and Destructors
- 16.11 Using Data Members and Member Functions
- 16.12 A Trap: Returning a Reference to a private Data Member
- 16.13 Assignment by Default Memberwise Copy
- 16.14 Software Reusability

Objectives

- To understand the software engineering concepts of **encapsulation** and data hiding
- To understand the notions of **data abstraction** and abstract data types (ADTs)
- To be able to create **C++ ADTs**, namely **classes**
- To understand how to **create, use, and destroy class objects**
- To be able to control access to object data members and member functions
- To begin to appreciate the value of **object orientation**

16.1 Introduction

- **Object-oriented programming (OOP)**
  - *Encapsulates* **data (attributes) and functions (behavior) into** packages called *classes*
  - Data and functions closely related!

- **Information hiding**
  - Implementation details are hidden within the classes themselves

- **Unit of C++ programming: the class**
  - A class is like a blueprint – reusable
  - Objects are *instantiated* (created) from the class
  - For example, a house is an instance of a "blueprint class"
  - C programmers concentrate on functions

16.2 Implementing a Time Abstract Data Type with a Class

- **Classes**
  - Model objects that have attributes (data members) and behaviors (member functions)
  - Defined using keyword `class`

Public: and Private: are member-access specifiers

`Time` is the constructor

`setTime`, `printMilitary`, and `printStandard` are member functions.

`hour`, `minute`, and `second` are data members

```
1  class Time {
2  public:
3      Time();
4      void setTime( int, int, int );
5      void printMilitary();
6      void printStandard();
7  private:
8      int hour;    // 0 - 23
9      int minute;  // 0 - 59
10     int second;  // 0 - 59
11 }; // end class Time
```

- **Format**
  - Body delineated with braces (`{` and `}`)
  - Class definition terminates with a semicolon

16.2 Implementing a Time Abstract Data Type with a Class (II)

Member functions and data

`Public` - accessible wherever the program has access to an object of class `Time`

`Private` - accessible only to member functions of the class

`Protected` - discussed later...

- **Constructor**
  - Special member function that initializes data members of a class object
  - Constructors cannot return values
  - Same name as the class

- **Definitions**
  - Once class defined, can be used as a data type

```
Time sunset,               // object of type Time
    arrayOfTimes[ 5 ],     // array of Time objects
    *pointerToTime,        // pointer to a Time object
    &dinnerTime = sunset;  // reference to a Time object
```

Note: The class name becomes the new type specifier

- **Binary scope resolution operator (∷)**
  - Specifies which class owns the member function
  - Different classes can have the same name for member functions

- **Format for definition class member functions:**
  ReturnType   *ClassName*∷*MemberFunctionName( )*   {   ...   }

- **If member function is defined *inside* the class**
  - Scope resolution operator and class name are not needed
  - Defining a function outside a class does not change it being `public` or `private`

- **Classes encourage software reuse**
  - Inheritance allows new classes to be derived from old ones

- **In following program**
  - `Time` constructor initializes the data members to `0`
    - Ensures that the object is in a consistent state when it is created

```cpp
1   // Fig. 16.2: fig16_02.cpp
2   // Time class.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   // Time abstract data type (ADT) definition
9   class Time {
10  public:
11     Time();                      // constructor
12     void setTime( int, int, int ); // set hour, minute, second
13     void printMilitary();        // print military time format
14     void printStandard();        // print standard time format
15  private:
16     int hour;      // 0 - 23
17     int minute;    // 0 - 59
18     int second;    // 0 - 59
19  }; // end class Time
20
21  // Time constructor initializes each data member to zero.
22  // Ensures all Time objects start in a consistent state.
23  Time::Time() { hour = minute = second = 0; }
24
```

fig16_02.cpp (Part 1 of 3)

```cpp
25   // Set a new Time value using military time. Perform validity
26   // checks on the data values. Set invalid values to zero.
27   void Time::setTime( int h, int m, int s )
28   {
29       hour = ( h >= 0 && h < 24 ) ? h : 0;
30       minute = ( m >= 0 && m < 60 ) ? m : 0;
31       second = ( s >= 0 && s < 60 ) ? s : 0;
32   } // end function setTime
33
34   // Print Time in military format
35   void Time::printMilitary()
36   {
37       cout << ( hour < 10 ? "0" : "" ) << hour << ":"
38            << ( minute < 10 ? "0" : "" ) << minute;
39   } // end function printMilitary
40
41   // Print Time in standard format
42   void Time::printStandard()
43   {
44       cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
45            << ":" << ( minute < 10 ? "0" : "" ) << minute
46            << ":" << ( second < 10 ? "0" : "" ) << second
47            << ( hour < 12 ? " AM" : " PM" );
48   } // end function printStandard
49
```

fig16_02.cpp (Part 2 of 3)

```cpp
50   // Driver to test simple class Time
51   int main()
52   {
53       Time t;   // instantiate object t of class Time
54
55       cout << "The initial military time is ";
56       t.printMilitary();
57       cout << "\nThe initial standard time is ";
58       t.printStandard();
59
60       t.setTime( 13, 27, 6 );
61       cout << "\n\nMilitary time after setTime is ";
62       t.printMilitary();
63       cout << "\nStandard time after setTime is ";
64       t.printStandard();
65
66       t.setTime( 99, 99, 99 );   // attempt invalid settings
67       cout << "\n\nAfter attempting invalid settings:"
68            << "\nMilitary time: ";
69       t.printMilitary();
70       cout << "\nStandard time: ";
71       t.printStandard();
72       cout << endl;
73       return 0;
74   } // end function main
```

fig16_02.cpp (Part 3 of 3)

```
The initial military time is 00:00
The initial standard time is 12:00:00 AM

Military time after setTime is 13:27
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Military time: 00:00
Standard time: 12:00:00 AM
```

Program Output

16.3 Class Scope and Accessing Class Members

- **Class scope**
  - Data members and member functions
- **File scope**
  - Nonmember functions
- **Function scope**
  - Variables defined in member functions, destroyed after function completes
- **Inside a scope**
  - Members accessible by all member functions
  - Referenced by name
- **Outside a scope**
  - Use handles
    - An object name, a reference to an object or a pointer to an object
- **Accessing class members**
  - Same as `struct`s
  - Dot (`.`) for objects and arrow (`->`) for pointers
  - Example: `t.hour` is the hour element of `t`
  - `TimePtr->hour` is the hour element

```
1   // Fig. 16.3: fig16_03.cpp
2   // Demonstrating the class member access operators . and ->
3   //
4   // CAUTION: IN FUTURE EXAMPLES WE AVOID PUBLIC DATA!
5   #include <iostream>
6
7   using std::cout;
8   using std::endl;
9
10  // Simple class Count
11  class Count {
12  public:
13      int x;
14      void print() { cout << x << endl; }
15  }; // end class Count
16
17  int main()
18  {
19      Count counter,                   // create counter object
20            *counterPtr = &counter; // pointer to counter
21
22
23      cout << "Assign 7 to x and print using the object's name: ";
24      counter.x = 7;          // assign 7 to data member x
25      counter.print();        // call member function print
26
```

fig16_03.cpp (Part 1 of 2)

```
27      cout << "Assign 10 to x and print using a pointer: ";
28      counterPtr->x = 10;   // assign 10 to data member x
29      counterPtr->print(); // call member function print
30      return 0;
31  } // end function main
```

fig16_03.cpp (Part 2 of 2)

```
Assign 7 to x and print using the object's name: 7
Assign 10 to x and print using a pointer: 10
```

Program Output

- **Separating interface from implementation**
  - Easier to modify programs
  - C++ programs can be split into
    - *Header files* – **contains class definitions and function prototypes**
    - *Source-code files* – **contains member function definitions**

- **Program Outline:**
  - Using the same `Time` class as before, create a header file
  - Create a source code file
    - Load the header file to get the class definitions
    - Define the member functions of the class

```
1  // Fig. 16.4: time1.h
2  // Declaration of the Time class.
3  // Member functions are defined in time1.cpp
4
5  // prevent multiple inclusions of header file
6  #ifndef TIME1_H
7  #define TIME1_H
8
9  // Time abstract data type definition
10 class Time {
11 public:
12    Time();                      // constructor
13    void setTime( int, int, int ); // set hour, minute, second
14    void printMilitary();        // print military time format
15    void printStandard();        // print standard time format
16 private:
17    int hour;    // 0 - 23
18    int minute;  // 0 - 59
19    int second;  // 0 - 59
20 }; // end class Time
21
22 #endif
```

time1.h

```cpp
23  // Fig. 16.4: time1.cpp
24  // Member function definitions for Time class.
25  #include <iostream>
26
27  using std::cout;
28
29  #include "time1.h"
30
31  // Time constructor initializes each data member to zero.
32  // Ensures all Time objects start in a consistent state.
33  Time::Time() { hour = minute = second = 0; }
34
35  // Set a new Time value using military time. Perform validity
36  // checks on the data values. Set invalid values to zero.
37  void Time::setTime( int h, int m, int s )
38  {
39     hour   = ( h >= 0 && h < 24 ) ? h : 0;
40     minute = ( m >= 0 && m < 60 ) ? m : 0;
41     second = ( s >= 0 && s < 60 ) ? s : 0;
42  } // end function setTime
43
```

time1.cpp (Part 1 of 2)

```cpp
44  // Print Time in military format
45  void Time::printMilitary()
46  {
47     cout << ( hour < 10 ? "0" : "" ) << hour << ":"
48          << ( minute < 10 ? "0" : "" ) << minute;
49  } // end function printMilitary
50
51  // Print time in standard format
52  void Time::printStandard()
53  {
54     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
55          << ":" << ( minute < 10 ? "0" : "" ) << minute
56          << ":" << ( second < 10 ? "0" : "" ) << second
57          << ( hour < 12 ? " AM" : " PM" );
58  } // end function printStandard
```

time1.cpp (Part 2 of 2)

```cpp
59  // Fig. 16.4: fig16_04.cpp
60  // Driver for Time1 class
61  // NOTE: Compile with time1.cpp
62  #include <iostream>
63
64  using std::cout;
65  using std::endl;
66
67  #include "time1.h"
68
69  // Driver to test simple class Time
70  int main()
71  {
72     Time t;   // instantiate object t of class time
73
74     cout << "The initial military time is ";
75     t.printMilitary();
76     cout << "\nThe initial standard time is ";
77     t.printStandard();
78
79     t.setTime( 13, 27, 6 );
80     cout << "\n\nMilitary time after setTime is ";
81     t.printMilitary();
82     cout << "\nStandard time after setTime is ";
83     t.printStandard();
84
```

fig16_04.cpp (Part 1 of 2)

```cpp
85     t.setTime( 99, 99, 99 );   // attempt invalid settings
86     cout << "\n\nAfter attempting invalid settings:\n"
87          << "Military time: ";
88     t.printMilitary();
89     cout << "\nStandard time: ";
90     t.printStandard();
91     cout << endl;
92     return 0;
93  } // end function main
```

fig16_04.cpp (Part 2 of 2)

Program Output

```
The initial military time is 00:00
The initial standard time is 12:00:00 AM

Military time after setTime is 13:27
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Military time: 00:00
Standard time: 12:00:00 AM
```

16.5 Controlling Access to Members

- **Purpose of** `public`
  - Give clients a view of the *services* the class provides (interface)

- **Purpose of** `private`
  - Default setting
  - Hide details of how the class accomplishes its tasks (implementation)
  - `Private` members only accessible through the `public` interface using `public` member functions

```cpp
1  // Fig. 16.5: fig16_05.cpp
2  // Demonstrate errors resulting from attempts
3  // to access private class members.
4  #include <iostream>
5
6  using std::cout;
7
8  #include "time1.h"
9
10 int main()
11 {
12    Time t;
13
14    // Error: 'Time::hour' is not accessible
15    t.hour = 7;
16
17    // Error: 'Time::minute' is not accessible
18    cout << "minute = " << t.minute;
19
20    return 0;
21 } // end function main
```

fig16_05.cpp

```
Borland C++ command-line compiler error messages

Time1.cpp:
Fig16_05.cpp:
Error E2247 Fig16_05.cpp 15:
    'Time::hour' is not accessible in function main()
Error E2247 Fig16_05.cpp 18:
    'Time::minute' is not accessible in function main()

*** 2 errors in Compile ***
```

```
Microsoft Visual C++ compiler error messages

Compiling...
Fig16_05.cpp
D:\Fig16_05.cpp(15) : error C2248: 'hour' : cannot access private
member declared in class 'Time'
D:\Fig16_05\time1.h(18) : see declaration of 'hour'
D:\Fig16_05.cpp(18) : error C2248: 'minute' : cannot access private
member declared in class 'Time'
D:\time1.h(19) : see declaration of 'minute'
Error executing cl.exe.

test.exe - 2 error(s), 0 warning(s)
```

16.6 Access Functions and Utility Functions

- **Utility functions**
  - `private` functions that support the operation of public functions
  - Not intended to be used directly by clients

- **Access functions**
  - `public` functions that read/display data or check conditions
  - For a container, it could call the `isEmpty` function

- **Next**
  - Program to take in monthly sales and output the total
  - Implementation not shown, only access functions

```cpp
1   // Fig. 16.6: salesp.h
2   // SalesPerson class definition
3   // Member functions defined in salesp.cpp
4   #ifndef SALESP_H
5   #define SALESP_H
6
7   class SalesPerson {
8   public:
9      SalesPerson();                // constructor
10     void getSalesFromUser();      // get sales figures from keyboard
11     void setSales( int, double ); // User supplies one month's
12                                   // sales figures.
13     void printAnnualSales();
14
15  private:
16     double totalAnnualSales();    // utility function
17     double sales[ 12 ];           // 12 monthly sales figures
18  }; // end class SalesPerson
19
20  #endif
```

salesp.h

```cpp
21  // Fig. 16.6: salesp.cpp
22  // Member functions for class SalesPerson
23  #include <iostream>
24
25  using std::cout;
26  using std::cin;
27  using std::endl;
28
29  #include <iomanip>
30
31  using std::setprecision;
32  using std::setiosflags;
33  using std::ios;
34
35  #include "salesp.h"
36
37  // Constructor function initializes array
38  SalesPerson::SalesPerson()
39  {
40     for ( int i = 0; i < 12; i++ )
41        sales[ i ] = 0.0;
42  } // end SalesPerson constructor
43
```

salesp.cpp (Part 1 of 3)

```cpp
44  // Function to get 12 sales figures from the user
45  // at the keyboard
46  void SalesPerson::getSalesFromUser()
47  {
48     double salesFigure;
49
50     for ( int i = 1; i <= 12; i++ ) {
51        cout << "Enter sales amount for month " << i << ": ";
52
53        cin >> salesFigure;
54        setSales( i, salesFigure );
55     } // end for
56  } // end function getSalesFromUser
57
58  // Function to set one of the 12 monthly sales figures.
59  // Note that the month value must be from 0 to 11.
60  void SalesPerson::setSales( int month, double amount )
61  {
62     if ( month >= 1 && month <= 12 && amount > 0 )
63        sales[ month - 1 ] = amount; // adjust for subscripts 0-11
64     else
65        cout << "Invalid month or sales figure" << endl;
66  } // end function setSales
67
```

salesp.cpp (Part 2 of 3)

```cpp
68  // Print the total annual sales
69  void SalesPerson::printAnnualSales()
70  {
71     cout << setprecision( 2 )
72          << setiosflags( ios::fixed | ios::showpoint )
73          << "\nThe total annual sales are: $"
74          << totalAnnualSales() << endl;
75  } // end function printAnnualSales
76
77  // Private utility function to total annual sales
78  double SalesPerson::totalAnnualSales()
79  {
80     double total = 0.0;
81
82     for ( int i = 0; i < 12; i++ )
83        total += sales[ i ];
84
85     return total;
86  } // end function totalAnnualSales
```

salesp.cpp (Part 3 of 3)

```
87  // Fig. 16.6: fig16_06.cpp
88  // Demonstrating a utility function
89  // Compile with salesp.cpp
90  #include "salesp.h"
91
92  int main()
93  {
94     SalesPerson s;          // create SalesPerson object s
95
96     s.getSalesFromUser();   // note simple sequential code
97     s.printAnnualSales();   // no control structures in main
98     return 0;
99  } // end function main
```

fig16_06.cpp

```
Enter sales amount for month 1: 5314.76
Enter sales amount for month 2: 4292.38
Enter sales amount for month 3: 4589.83
Enter sales amount for month 4: 5534.03
Enter sales amount for month 5: 4376.34
Enter sales amount for month 6: 5698.45
Enter sales amount for month 7: 4439.22
Enter sales amount for month 8: 5893.57
Enter sales amount for month 9: 4909.67
Enter sales amount for month 10: 5123.45
Enter sales amount for month 11: 4024.97
Enter sales amount for month 12: 5923.92

The total annual sales are: $60120.59
```

Program Output

16.7 Initializing Class Objects: Constructors

- **Constructor function**
  - Can initialize class members
  - Same name as the class, no return type
  - Member variables can be initialized by the constructor or set afterwards

- **Defining objects**
  - Initializers can be provided
  - Initializers passed as arguments to the class' constructor

- **Format**

  Type   ObjectName(value1, value2, …) **;**
  - Constructor assigns *value1, value2,* etc. to its member variables
  - If not enough values specified, rightmost parameters set to their default (specified by programmer)

  ```
  myClass myObject(3, 4.0);
  ```

16.8 Using Default Arguments with Constructors

- **Default constructor**
  - One per class
  - Can be invoked without arguments
  - Has default arguments

- **Default arguments**
  - Set in default constructor function prototype (in class definition)
    - Do not set defaults in the function definition, outside of a class
  - Example:
    ```
    SampleClass( int = 0, float = 0);
    ```
    - Constructor has same name as class

```cpp
1  // Fig. 16.7: time2.h
2  // Declaration of the Time class.
3  // Member functions are defined in time2.cpp
4
5  // preprocessor directives that
6  // prevent multiple inclusions of header file
7  #ifndef TIME2_H
8  #define TIME2_H
9
10 // Time abstract data type definition
11 class Time {
12 public:
13    Time( int = 0, int = 0, int = 0 );  // default constructor
14    void setTime( int, int, int ); // set hour, minute, second
15    void printMilitary();          // print military time format
16    void printStandard();          // print standard time format
17 private:
18    int hour;     // 0 - 23
19    int minute;   // 0 - 59
20    int second;   // 0 - 59
21 }; // end class Time
22
23 #endif
```

time2.h

```cpp
24  // Fig. 16.7: time2.cpp
25  // Member function definitions for Time class.
26  #include <iostream>
27
28  using std::cout;
29
30  #include "time2.h"
31
32  // Time constructor initializes each data member to zero.
33  // Ensures all Time objects start in a consistent state.
34  Time::Time( int hr, int min, int sec )
35     { setTime( hr, min, sec ); }
36
37  // Set a new Time value using military time. Perform validity
38  // checks on the data values. Set invalid values to zero.
39  void Time::setTime( int h, int m, int s )
40  {
41     hour   = ( h >= 0 && h < 24 ) ? h : 0;
42     minute = ( m >= 0 && m < 60 ) ? m : 0;
43     second = ( s >= 0 && s < 60 ) ? s : 0;
44  } // end function setTime
45
```

time2.cpp (Part 1 of 2)

```cpp
46  // Print Time in military format
47  void Time::printMilitary()
48  {
49     cout << ( hour < 10 ? "0" : "" ) << hour << ":"
50          << ( minute < 10 ? "0" : "" ) << minute;
51  } // end function printMilitary
52
53  // Print Time in standard format
54  void Time::printStandard()
55  {
56     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
57          << ":" << ( minute < 10 ? "0" : "" ) << minute
58          << ":" << ( second < 10 ? "0" : "" ) << second
59          << ( hour < 12 ? " AM" : " PM" );
60  } // end function printStandard
```

time2.cpp (Part 2 of 2)

```
61  // Fig. 16.7: fig16_07.cpp
62  // Demonstrating a default constructor
63  // function for class Time.
64  #include <iostream>
65
66  using std::cout;
67  using std::endl;
68
69  #include "time2.h"
70
71  int main()
72  {
73     Time t1,                 // all arguments defaulted
74          t2( 2 ),            // minute and second defaulted
75          t3( 21, 34 ),       // second defaulted
76          t4( 12, 25, 42 ), // all values specified
77          t5( 27, 74, 99 ); // all bad values specified
78
79     cout << "Constructed with:\n"
80          << "all arguments defaulted:\n   ";
81     t1.printMilitary();
82     cout << "\n   ";
83     t1.printStandard();
84
```

fig16_07.cpp (Part 1 of 2)

```
85     cout << "\nhour specified; minute and second defaulted:"
86          << "\n   ";
87     t2.printMilitary();
88     cout << "\n   ";
89     t2.printStandard();
90
91     cout << "\nhour and minute specified; second defaulted:"
92          << "\n   ";
93     t3.printMilitary();
94     cout << "\n   ";
95     t3.printStandard();
96
97     cout << "\nhour, minute, and second specified:"
98          << "\n   ";
99     t4.printMilitary();
100    cout << "\n   ";
101    t4.printStandard();
102
103    cout << "\nall invalid values specified:"
104         << "\n   ";
105    t5.printMilitary();
106    cout << "\n   ";
107    t5.printStandard();
108    cout << endl;
109
110    return 0;
111 } // end function main
```

fig16_07.cpp (Part 2 of 2)

```
Constructed with:
all arguments defaulted:
    00:00
    12:00:00 AM
hour specified; minute and second defaulted:
    02:00
    2:00:00 AM
hour and minute specified; second defaulted:
    21:34
    9:34:00 PM
hour, minute, and second specified:
    12:25
    12:25:42 PM
all invalid values specified:
    00:00
    12:00:00 AM
```

Program Output

16.9 Using Destructors

- **Destructor**
  - Member function of class
  - Performs termination housekeeping before the system reclaims the object's memory
  - Complement of the constructor
  - Name is *tilde* (~) followed by the class name
    - `~Time`
    - Recall that the constructor's name is the class name
  - Receives no parameters, returns no value
  - One destructor per class - no overloading allowed

16.10 Calling Constructors and Destructors

- **Constructors and destructors called automatically**
  - Order depends on scope of objects
- **Global scope objects**
  - Constructors called before any other function (including `main`)
  - Destructors called when `main` terminates (or `exit` function called)
  - Destructors not called if program terminates with `abort`
- **Automatic local objects**
  - Constructors called when objects defined
  - Destructors called when objects leave scope (when the block in which they are defined is exited)
  - Destructors not called if program ends with `exit` or `abort`
- **`static` local objects**
  - Constructors called when execution reaches the point where the objects are defined
  - Destructors called when `main` terminates or the `exit` function is called
  - Destructors not called if the program ends with `abort`

```cpp
1  // Fig. 16.8: create.h
2  // Definition of class CreateAndDestroy.
3  // Member functions defined in create.cpp.
4  #ifndef CREATE_H
5  #define CREATE_H
6
7  class CreateAndDestroy {
8  public:
9     CreateAndDestroy( int );  // constructor
10    ~CreateAndDestroy();      // destructor
11 private:
12    int data;
13 }; // end class CreateAndDestroy
14
15 #endif
```

create.h

```cpp
16  // Fig. 16.8: create.cpp
17  // Member function definitions for class CreateAndDestroy
18  #include <iostream>
19
20  using std::cout;
21  using std::endl;
22
23  #include "create.h"
24
25  CreateAndDestroy::CreateAndDestroy( int value )
26  {
27     data = value;
28     cout << "Object " << data << "    constructor";
29  } // end CreateAndDestroy constructor
30
31  CreateAndDestroy::~CreateAndDestroy()
32     { cout << "Object " << data << "    destructor " << endl; }
```

create.cpp

```cpp
33  // Fig. 16.8: fig16_08.cpp
34  // Demonstrating the order in which constructors and
35  // destructors are called.
36  #include <iostream>
37
38  using std::cout;
39  using std::endl;
40
41  #include "create.h"
42
43  void create( void );    // prototype
44
45  CreateAndDestroy first( 1 );   // global object
46
47  int main()
48  {
49     cout << "    (global created before main)" << endl;
50
51     CreateAndDestroy second( 2 );          // local object
52     cout << "    (local automatic in main)" << endl;
53
54     static CreateAndDestroy third( 3 );   // local object
55     cout << "    (local static in main)" << endl;
56
57     create();  // call function to create objects
58
```

fig16_08.cpp (Part 1 of 2)

```
59      CreateAndDestroy fourth( 4 );        // local object
60      cout << "    (local automatic in main)" << endl;
61      return 0;
62  } // end function main
63
64  // Function to create objects
65  void create( void )
66  {
67      CreateAndDestroy fifth( 5 );
68      cout << "    (local automatic in create)" << endl;
69
70      static CreateAndDestroy sixth( 6 );
71      cout << "    (local static in create)" << endl;
72
73      CreateAndDestroy seventh( 7 );
74      cout << "    (local automatic in create)" << endl;
75  } // end function create
```

fig16_08.cpp (Part 2 of 2)

```
Object 1    constructor    (global created before main)
Object 2    constructor    (local automatic in main)
Object 3    constructor    (local static in main)
Object 5    constructor    (local automatic in create)
Object 6    constructor    (local static in create)
Object 7    constructor    (local automatic in create)
Object 7    destructor
Object 5    destructor
Object 4    constructor    (local automatic in main)
Object 4    destructor
Object 2    destructor
Object 6    destructor
Object 3    destructor
Object 1    destructor
```

Program Output

16.11 Using Data Members and Member Functions

- **Classes provide `public` member functions**
  - Set (i.e., write) or *get* (i.e., read) values of `private` data members
  - Adjustment of bank balance (a `private` data member of class `BankAccount`) by member function `computeInterest`

- **Naming**
  - Member function that *set*s `interestRate` typically named `setInterestRate`
  - Member function that *get*s `interestRate` would typically be called `getInterestRate`

16.11 Using Data Members and Member Functions (II)

- **Do *set* and *get* capabilities effectively make data members `public`?**
  - No!
  - Programmer decides what the function can set and what information the function can get

- **`public` set functions should**
  - Check attempts to modify data members
  - Ensure that the new value is appropriate for that data item
  - Example: an attempt to *set* the day of the month to 37 would be rejected
  - Programmer must include these features

```
1   // Fig. 16.9: time3.h
2   // Declaration of the Time class.
3   // Member functions defined in time3.cpp
4
5   // preprocessor directives that
6   // prevent multiple inclusions of header file
7   #ifndef TIME3_H
8   #define TIME3_H
9
10  class Time {
11  public:
12     Time( int = 0, int = 0, int = 0 );   // constructor
13
14     // set functions
15     void setTime( int, int, int );   // set hour, minute, second
16     void setHour( int );    // set hour
17     void setMinute( int );  // set minute
18     void setSecond( int );  // set second
19
20     // get functions
21     int getHour();          // return hour
22     int getMinute();        // return minute
23     int getSecond();        // return second
24
```

```
25     void printMilitary();   // output military time
26     void printStandard();   // output standard time
27
28  private:
29     int hour;               // 0 - 23
30     int minute;             // 0 - 59
31     int second;             // 0 - 59
32  }; // end class Time
33
34  #endif
```

```cpp
35  // Fig. 16.9: time3.cpp
36  // Member function definitions for Time class.
37  #include <iostream>
38
39  using std::cout;
40
41  #include "time3.h"
42
43  // Constructor function to initialize private data.
44  // Calls member function setTime to set variables.
45  // Default values are 0 (see class definition).
46  Time::Time( int hr, int min, int sec )
47     { setTime( hr, min, sec ); }
48
49  // Set the values of hour, minute, and second.
50  void Time::setTime( int h, int m, int s )
51  {
52     setHour( h );
53     setMinute( m );
54     setSecond( s );
55  } // end function setTime
56
57  // Set the hour value
58  void Time::setHour( int h )
59     { hour = ( h >= 0 && h < 24 ) ? h : 0; }
60
```

```cpp
61  // Set the minute value
62  void Time::setMinute( int m )
63     { minute = ( m >= 0 && m < 60 ) ? m : 0; }
64
65  // Set the second value
66  void Time::setSecond( int s )
67     { second = ( s >= 0 && s < 60 ) ? s : 0; }
68
69  // Get the hour value
70  int Time::getHour() { return hour; }
71
72  // Get the minute value
73  int Time::getMinute() { return minute; }
74
75  // Get the second value
76  int Time::getSecond() { return second; }
77
78  // Print time is military format
79  void Time::printMilitary()
80  {
81     cout << ( hour < 10 ? "0" : "" ) << hour << ":"
82          << ( minute < 10 ? "0" : "" ) << minute;
83  } // end function printMilitary
84
```

```
85  // Print time in standard format
86  void Time::printStandard()
87  {
88      cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
89          << ":" << ( minute < 10 ? "0" : "" ) << minute
90          << ":" << ( second < 10 ? "0" : "" ) << second
91          << ( hour < 12 ? " AM" : " PM" );
92  } // end function printStandard
```

time3.cpp (Part 3 of 3)

```
93   // Fig. 16.9: fig16_09.cpp
94   // Demonstrating the Time class set and get functions
95   #include <iostream>
96
97   using std::cout;
98   using std::endl;
99
100  #include "time3.h"
101
102  void incrementMinutes( Time &, const int );
103
104  int main()
105  {
106      Time t;
107
108      t.setHour( 17 );
109      t.setMinute( 34 );
110      t.setSecond( 25 );
111
```

```
112      cout << "Result of setting all valid values:\n"
113          << "  Hour: " << t.getHour()
114          << "  Minute: " << t.getMinute()
115          << "  Second: " << t.getSecond();
116
117      t.setHour( 234 );    // invalid hour set to 0
118      t.setMinute( 43 );
119      t.setSecond( 6373 ); // invalid second set to 0
120
121      cout << "\n\nResult of attempting to set invalid hour and"
122          << " second:\n  Hour: " << t.getHour()
123          << "  Minute: " << t.getMinute()
124          << "  Second: " << t.getSecond() << "\n\n";
125
126      t.setTime( 11, 58, 0 );
127      incrementMinutes( t, 3 );
128
129      return 0;
130  } // end function main
131
132  void incrementMinutes( Time &tt, const int count )
133  {
134      cout << "Incrementing minute " << count
135          << " times:\nStart time: ";
136      tt.printStandard();
137
```

fig16_09.cpp (Part 1 of 2)

```
138     for ( int i = 0; i < count; i++ ) {
139         tt.setMinute( ( tt.getMinute() + 1 ) % 60 );
140
141         if ( tt.getMinute() == 0 )
142             tt.setHour( ( tt.getHour() + 1 ) % 24 );
143
144         cout << "\nminute + 1: ";
145         tt.printStandard();
146     } // end for
147
148     cout << endl;
149 } // end function incrementMinutes
```

Program Output

```
Result of setting all valid values:
  Hour: 17  Minute: 34  Second: 25

Result of attempting to set invalid hour and second:
  Hour: 0  Minute: 43  Second: 0

Incrementing minute 3 times:
Start time: 11:58:00 AM
minute + 1: 11:59:00 AM
minute + 1: 12:00:00 PM
minute + 1: 12:01:00 PM
```

16.12 A Subtle Trap: Returning a Reference to a Private Data Member

- **Reference to an object**
  - Alias for the name of the object
  - May be used on the left side of an assignment statement
  - Reference can receive a value, which changes the original object as well

- **One way to use this capability (unfortunately!)**
  - Have a `public` member function of a class return a `non-const` reference to a `private` data member
  - This reference can be modified, which changes the original data

```
1   // Fig. 16.10: time4.h
2   // Declaration of the Time class.
3   // Member functions defined in time4.cpp
4
5   // preprocessor directives that
6   // prevent multiple inclusions of header file
7   #ifndef TIME4_H
8   #define TIME4_H
9
10  class Time {
11  public:
12     Time( int = 0, int = 0, int = 0 );
13     void setTime( int, int, int );
14     int getHour();
15     int &badSetHour( int );   // DANGEROUS reference return
16  private:
17     int hour;
18     int minute;
19     int second;
20  }; // end class Time
21
22  #endif
```

time4.h

```
23  // Fig. 16.10: time4.cpp
24  // Member function definitions for Time class.
25  #include "time4.h"
26
27  // Constructor function to initialize private data.
28  // Calls member function setTime to set variables.
29  // Default values are 0 (see class definition).
30  Time::Time( int hr, int min, int sec )
31     { setTime( hr, min, sec ); }
32
33  // Set the values of hour, minute, and second.
34  void Time::setTime( int h, int m, int s )
35  {
36     hour   = ( h >= 0 && h < 24 ) ? h : 0;
37     minute = ( m >= 0 && m < 60 ) ? m : 0;
38     second = ( s >= 0 && s < 60 ) ? s : 0;
39  } // end function setTime
40
41  // Get the hour value
42  int Time::getHour() { return hour; }
43
```

time4.cpp (Part 1 of 2)

```cpp
44   // POOR PROGRAMMING PRACTICE:
45   // Returning a reference to a private data member.
46   int &Time::badSetHour( int hh )
47   {
48       hour = ( hh >= 0 && hh < 24 ) ? hh : 0;
49
50       return hour;   // DANGEROUS reference return
51   } // end function badSetHour
```

time4.cpp

```cpp
52   // Fig. 16.10: fig16_10.cpp
53   // Demonstrating a public member function that
54   // returns a reference to a private data member.
55   // Time class has been trimmed for this example.
56   #include <iostream>
57
58   using std::cout;
59   using std::endl;
60
61   #include "time4.h"
62
63   int main()
64   {
65       Time t;
66       int &hourRef = t.badSetHour( 20 );
67
68       cout << "Hour before modification: " << hourRef;
69       hourRef = 30;   // modification with invalid value
70       cout << "\nHour after modification: " << t.getHour();
71
```

```cpp
72       // Dangerous: Function call that returns
73       // a reference can be used as an lvalue!
74       t.badSetHour( 12 ) = 74;
75       cout << "\n\n*******************************\n"
76            << "POOR PROGRAMMING PRACTICE!!!!!!!!\n"
77            << "badSetHour as an lvalue, Hour: "
78            << t.getHour()
79            << "\n*******************************" << endl;
80
81       return 0;
82   }
```

fig16_10.cpp

Program Output

```
Hour before modification: 20
Hour after modification: 30

*******************************
POOR PROGRAMMING PRACTICE!!!!!!!!
badSetHour as an lvalue, Hour: 74
*******************************
```

16.13 Assignment by Default Memberwise Copy

- **Assignment operator (=)**
    - Sets variables equal, i.e., `x = y`;
    - Can be used to assign an object to another object of the same type
    - Memberwise copy — member by member copy

        `myObject1 = myObject2`;

- **Objects may be**
    - Passed as function arguments
    - Returned from functions (call-by-value default)
        - Use pointers for call by reference

```cpp
1   // Fig. 16.11: fig16_11.cpp
2   // Demonstrating that class objects can be assigned
3   // to each other using default memberwise copy
4   #include <iostream>
5
6   using std::cout;
7   using std::endl;
8
9   // Simple Date class
10  class Date {
11  public:
12     Date( int = 1, int = 1, int = 1990 ); // default constructor
13     void print();
14  private:
15     int month;
16     int day;
17     int year;
18  }; // end class Date
19
20  // Simple Date constructor with no range checking
21  Date::Date( int m, int d, int y )
22  {
23     month = m;
24     day = d;
25     year = y;
26  } // end Date constructor
27
```

fig16_11.cpp (Part 1 of 2)

```
28  // Print the Date in the form mm-dd-yyyy
29  void Date::print()
30     { cout << month << '-' << day << '-' << year; }
31
32  int main()
33  {
34     Date date1( 7, 4, 1993 ), date2;   // d2 defaults to 1/1/90
35
36     cout << "date1 = ";
37     date1.print();
38     cout << "\ndate2 = ";
39     date2.print();
40
41     date2 = date1;    // assignment by default memberwise copy
42     cout << "\n\nAfter default memberwise copy, date2 = ";
43     date2.print();
44     cout << endl;
45
46     return 0;
47  } // end function main
```

Program Output

```
date1 = 7-4-1993
date2 = 1-1-1990

After default memberwise copy, date2 = 7-4-1993
```

16.14   Software Reusability

- **Object-oriented programmers**
  - Concentrate on implementing useful classes

- **Tremendous opportunity to capture and catalog classes**
  - Accessed by large segments of the programming community
  - Class libraries exist for this purpose

- **Software**
  - Constructed from existing, well-defined, carefully tested, portable, widely available components
  - Speeds development of powerful, high-quality software

Complementos de Programação de Computadores – Aula 2
Classes e Abstracção de Dados

Mestrado Integrado em Electrónica Industrial e Computadores

Luís Paulo Reis

lpreis1970@gmail.com / lpreis@dsi.uminho.pt

Professor Associado do Departamento de Sistemas de Informação, Escola de Engenharia, Universidade do Minho, Portugal

(Slides Baseados em P.Deitel e H.Deitel 2010 e Paulo Cortez, 2011)