C++ Classes
&
Object Oriented Programming

Classes and Objects

- Class Definitions and Objects
- Member Functions
- Data Members
  - Get and Set functions
  - Constructors

C++ Program Structure

- Typical C++ Programs consist of:–
  - A function main
  - One or more classes
    - Each containing data members and member functions.

© 2007 Pearson Ed -All rights reserved.

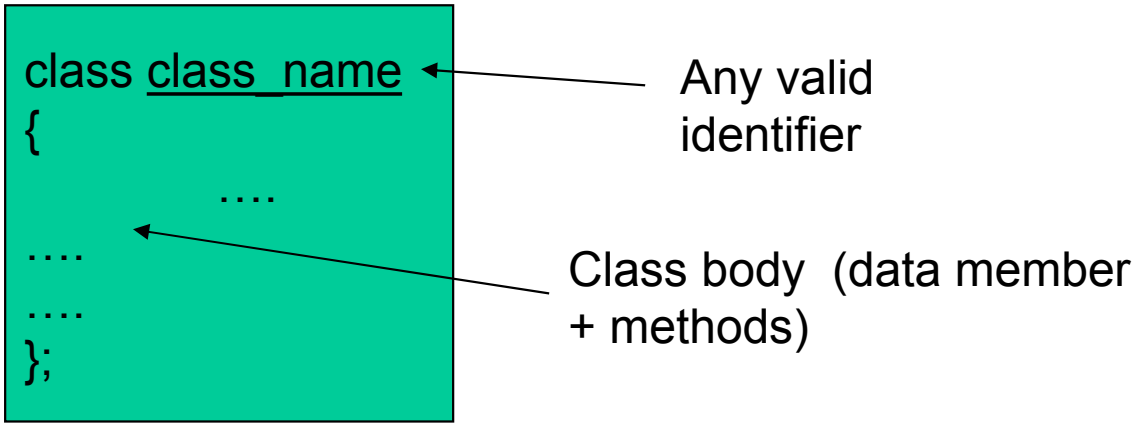
19.4 Defining a Class With a Member Function

- Class definition
  - Tells the compiler what member functions and data members belong to the class.
  - Keyword class followed by the class's name.
  - Class body is enclosed in braces ({})
    - Specifies data members and member functions
    - Access-specifier public:
      - Indicates that a member function or data member is accessible to other functions and member functions of other classes.

© 2007 Pearson Ed -All rights reserved.

Classes in C++

- A class definition begins with the keyword *class*.
- The body of the class is contained within a set of braces, { } ; (notice the semi-colon).

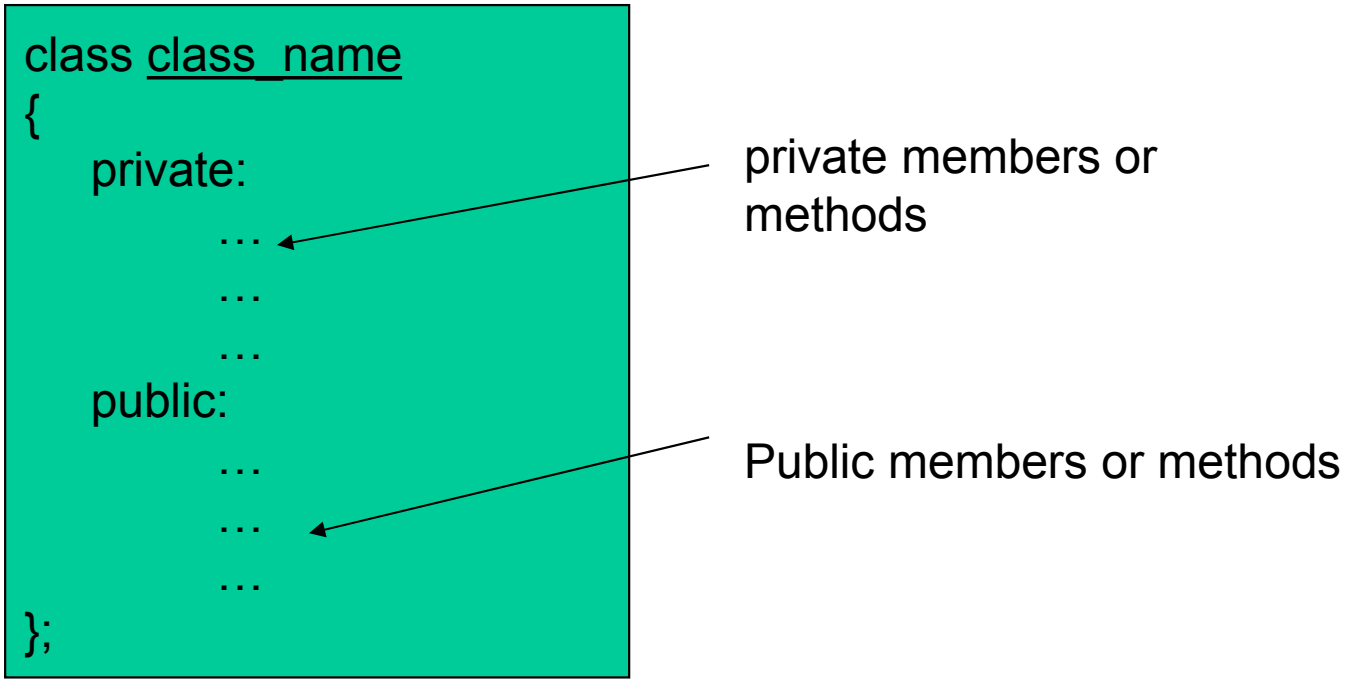


Classes in C++

- Within the body, the keywords *private:* and *public:* specify the access level of the members of the class.
  - the default is private.

- Usually, the data members of a class are declared in the *private:* section of the class and the member functions are in *public:* section.

Classes in C++


class class_name
{
private:
…
…
…
public:
…
…
…
};
private members or methods
Public members or methods

Classes in C++

- Member access specifiers
  - public:
    - can be accessed outside the class directly.
      - The public stuff is *the interface*.
  - private:
    - Accessible only to member functions of class
    - Private members and methods are for internal use only.

Class Example

- This class example shows how we can encapsulate (gather) a circle information into one package (unit or class)

```
class Circle
{
    private:
            double radius;
    public:
            void setRadius(double r);
            double getDiameter();
            double getArea();
            double getCircumference();
};
```

No need for others classes to access and retrieve its value directly. The class methods are responsible for that only.

They are accessible from outside the class, and they can access the member (radius)

Creating an object of a Class

- Declaring a variable of a class type creates an object. You can have many variables of the same type (class).
  - *Instantiation*
- Once an object of a certain class is instantiated, a new memory location is created for it to store its data members and code
- You can instantiate many objects from a class type.
  - Ex) Circle c; Circle *c;

Special Member Functions

- Constructor:
  - Public function member
  - called when a new object is created (instantiated).
  - Initialize data members.
  - Same name as class
  - No return type
  - Several constructors
    - Function overloading

Special Member Functions

```
class Circle
{
    private:
            double radius;
    public:
            Circle();
            Circle(int r);
             void setRadius(double r);
            double getDiameter();
            double getArea();
            double getCircumference();
};
```

Constructor with no argument

Constructor with one argument

Implementing class methods

- Class implementation: writing the code of class methods.
- There are two ways:

1. Member functions defined outside class
   - Using Binary scope resolution operator (`::`)
   - "Ties" member name to class name
   - Uniquely identify functions of particular class
   - Different classes can have member functions with same name

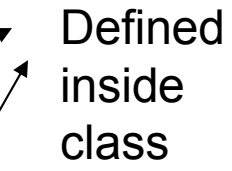
– Format for defining member functions

```
ReturnType ClassName::MemberFunctionName(
    ){
    ...
}
```

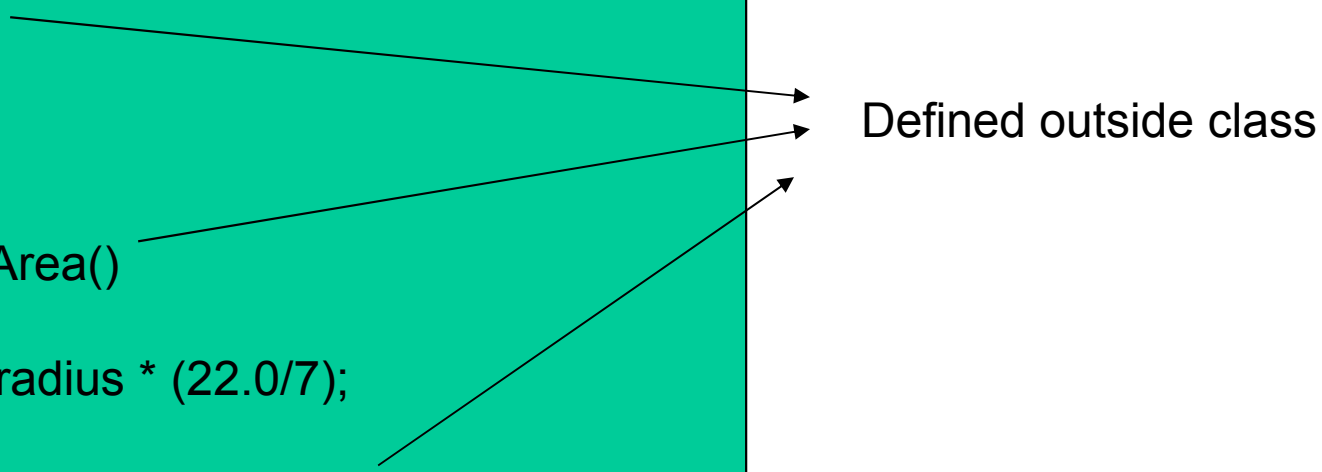
Implementing class methods

2. Member functions defined inside class
   - Do not need scope resolution operator, class name;

```
class Circle
{
    private:
            double radius;
    public:
            Circle() { radius = 0.0;}
            Circle(int r);
            void setRadius(double r){radius = r;}
            double getDiameter(){ return radius *2;}
            double getArea();
            double getCircumference();
};
```



```
class Circle
{
    private:
            double radius;
    public:
            Circle() { radius = 0.0;}
            Circle(int r);
            void setRadius(double r){radius = r;}
            double getDiameter(){ return radius *2;}
            double getArea();
            double getCircumference();
};
Circle::Circle(int r)
{
    radius = r;
}
double Circle::getArea()
{
    return radius * radius * (22.0/7);
}
double Circle:: getCircumference()
{
    return 2 * radius  * (22.0/7);
}
```



Accessing Class Members

- Operators to access class members
  - Dot member selection operator (.)
    - Object
    - Reference to object
  - Arrow member selection operator (->)
    - Pointers

```
class Circle
{
    private:
        double radius;
    public:
        Circle() { radius = 0.0;}
        Circle(int r);
        void setRadius(double r){radius = r;}
        double getDiameter(){ return radius *2;}
        double getArea();
        double getCircumference();
};
Circle::Circle(int r)
{
    radius = r;
}
double Circle::getArea()
{
    return radius * radius * (22.0/7);
}
double Circle:: getCircumference()
{
    return 2 * radius  * (22.0/7);
}
```

```
void main()
{
    Circle c1,c2(7);

    cout<<"The area of c1
        <<c1.getArea()<<"\n";

    //c1.raduis = 5;//syntax error
    c1.setRadius(5);

    cout<<"The circumference of c1:"
       << c1.getCircumference()<<"\n";

    cout<<"The Diameter of c2:"
       <<c2.getDiameter()<<"\n";
}
```

The second constructor is called

Since radius is a private class data member

```
class Circle
{
    private:
            double radius;
    public:
            Circle() { radius = 0.0;}
            Circle(int r);
            void setRadius(double r){radius = r;}
            double getDiameter(){ return radius *2;}
            double getArea();
            double getCircumference();
};
Circle::Circle(int r)
{
    radius = r;
}
double Circle::getArea()
{
    return radius * radius * (22.0/7);
}
double Circle:: getCircumference()
{
    return 2 * radius  * (22.0/7);
}
```

```
void main()
{
     Circle c(7);
     Circle *cp1 = &c;
     Circle *cp2 = new Circle(7);

     cout<<"The are of cp2:"
                 <<cp2->getArea();

}
```

Destructors

- Destructors
  - Special member function
  - Same name as class
    - Preceded with tilde (~)
  - No arguments
  - No return value
  - Cannot be overloaded
  - Before system reclaims object's memory
    - Reuse memory for new objects
    - Mainly used to de-allocate dynamic memory locations

Another class Example

- This class shows how to handle time parts.

```
class Time
{
    private:
        int *hour,*minute,*second;
    public:
        Time();
        Time(int h,int m,int s);
        void printTime();
        void setTime(int h,int m,int s);
        int getHour(){return *hour;}
        int getMinute(){return *minute;}
        int getSecond(){return *second;}
        void setHour(int h){*hour = h;}
        void setMinute(int m){*minute = m;}
        void setSecond(int s){*second = s;}
        ~Time();
};
```

Dynamic locations should be allocated to pointers first

```
Time::Time()
}
        hour = new int;
        minute = new int;
        second = new int;
        *hour = *minute = *second = 0;
{

Time::Time(int h,int m,int s)
}
        hour = new int;
        minute = new int;
        second = new int;
        *hour = h;
        *minute = m;
        *second = s;
{

void Time::setTime(int h,int m,int s)
}
        *hour = h;
        *minute = m;
        *second = s;
{
```

```
void Time::printTime()
}
     cout<<"The time is : ("<<*hour<<":"<<*minute<<":"<<*second<<")"
                 >>endl;
{

Time::~Time()
}
       delete hour; delete minute;delete second;
{

void main()
}
       Time *t;
       t= new Time(3,55,54);
       t->printTime();

       t->setHour(7);
       t->setMinute(17);
       t->setSecond(43);

       t->printTime();

       delete t;
{
```

Destructor: used here to de-allocate memory locations

Output:
The time is : (3:55:54)
The time is : (7:17:43)
Press any key to continue

When executed, the destructor is called

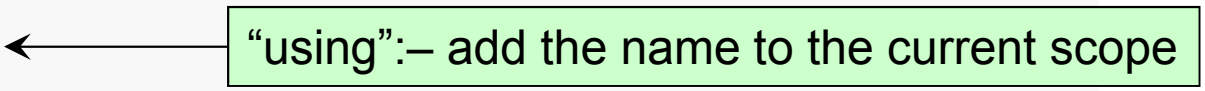
Reasons for OOP

1. Simplify programming
2. Interfaces
   - Information hiding:
     - Implementation details hidden within classes themselves
3. Software reuse
   - Class objects included as members of other classes

C++ Gradebook Example

Deitel & Deitel Fig 19.1

```
1  // Fig. 19.1: fig19_01.cpp
2  // Define class GradeBook with a member function displayMessage;
3  // Create a GradeBook object and call its displayMessage function.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  // GradeBook class definition
9  class GradeBook
10 {
11 public:
12    // function that displays a welcome message to the GradeBook user
13    void displayMessage()
14    {
15       cout << "Welcome to the Grade Book!" << endl;
16    } // end function displayMessage
17 }; // end class GradeBook
18
19 // function main begins program execution
20 int main()
21 {
22    GradeBook myGradeBook; // create a GradeBook object named myGradeBook
23    myGradeBook.displayMessage(); // call object's displayMessage function
24    return 0; // indicate successful termination
25 } // end main
```




```
Welcome to the Grade Book!
```

C++ Gradebook Example

```
1  // Fig. 19.1: fig19_01.cpp
2  // Define class GradeBook with a member function displayMessage;
3  // Create a GradeBook object and call its displayMessage function.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  // GradeBook class definition
9  class GradeBook
10 {
11 public:
12    // function that displays a welcome message to the GradeBook user
13    void displayMessage()
14    {
15       cout << "Welcome to the Grade Book!" << endl;
16    } // end function displayMessage
17 }; // end class GradeBook
18
19 // function main begins program execution
20 int main()
21 {
22    GradeBook myGradeBook; // create a GradeBook object named myGradeBook
23    myGradeBook.displayMessage(); // call object's displayMessage function
24    return 0; // indicate successful termination
25 } // end main
```

Beginning of class definition for class `GradeBook`

Beginning of class body

Access specifier `public`; makes members available to the public

Member function `displayMessage` returns nothing

End of class body

Use dot operator to call `GradeBook`'s member function

```
Welcome to the Grade Book!
```

© 2007 Pearson Ed -All rights reserved.

C++ Gradebook Example

```
1  // Fig. 19.1: fig19_01.cpp
2  // Define class GradeBook with a member function displayMessage;
3  // Create a GradeBook object and call its displayMessage function.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  // GradeBook class definition
9  class GradeBook
10 {
11 public:
12    // function that displays a welcome message to the GradeBook user
13    void displayMessage()
14    {
15       cout << "Welcome to the Grade Book!" << endl;
16    } // end function displayMessage
17 }; // end class GradeBook
18
19 // function main begins program exe
20 int main()
21 {
22    GradeBook myGradeBook; // create a GradeBook object named myGradeBook
23    myGradeBook.displayMessage(); // call object's displayMessage function
24    return 0; // indicate successful termination
25 } // end main
```

```
Welcome to the Grade Book!
```

Declaring an object of this class as an automatic variable of **`main`**

I.e., allocated on The Stack

Member Functions with Parameters

```
1  // Fig. 19.3: fig19_03.cpp
2  // Define class GradeBook with a member function that takes a parameter;
3  // Create a GradeBook object and call its displayMessage function.
4  #include <iostream>
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  #include <string> // program uses C++ standard string class
10 using std::string;
11 using std::getline;
12
13 // GradeBook class definition
14 class GradeBook
15 {
16 public:
17    // function that displays a welcome message to the GradeBook user
18    void displayMessage( string courseName )
19    {
20       cout << "Welcome to the grade book for\n" << courseName << "!"
21          << endl;
22    } // end function displayMessage
23 }; // end class GradeBook
24
25 // function main begins program execution
26 int main()
27 {
28    string nameOfCourse; // string of characters to store the course name
29    GradeBook myGradeBook; // create a GradeBook object named myGradeBook
30
```

Include `string` class definition

Member function parameter

Use the function parameter as a variable

© 2007 Pearson Ed -All rights reserved.

Member Functions with Parameters (cont.)

```
31    // prompt for and input course name
32    cout << "Please enter the course name:" << endl;
33    getline( cin, nameOfCourse ); // read a course name with blanks
34    cout << endl; // output a blank line
35
36    // call myGradeBook's displayMessage function
37    // and pass nameOfCourse as an argument
38    myGradeBook.displayMessage( nameOfCourse );
39    return 0; // indicate successful termination
40 } // end main
```

```
Please enter the course name:
CS101 Introduction to C++ Programming

Welcome to the grade book for
CS101 Introduction to C++ Programming!
```

© 2007 Pearson Ed -All rights reserved.

Useful Tidbits

- A **`string`**
  - Represents a string of characters.
  - An object of C++ Standard Library class std::string
    - Defined in header file <string>.
  - *Not* a character array as in *C*.

- Library function **`getline`**
  - Used to retrieve input until newline is encountered
  - Example
    - getline(cin, nameOfCourse);
      - Inputs a line from standard input into string object nameOfCourse.
    - Defined in header file <iostream>.

© 2007 Pearson Ed -All rights reserved.

Data Members of a Class

- Declared in the body of the class
- May be *public* or *private*
- Exist throughout the life of the object.
- Stored in class object.
- Each object has its own copy.
    - May be objects of any type

© 2007 Pearson Ed -All rights reserved.

Access-specifier *private*

- Makes any member accessible only to *member functions* of the class.
  - May be applied to *data members* and *member functions*
- Default access for class members
  - Encourages “information hiding”

Public and Private Members

```
1  // Fig. 19.5: fig19_05.cpp
2  // Define class GradeBook that contains a courseName data member
3  // and member functions to set and get its value;
4  // Create and manipulate a GradeBook object with these functions.
5  #include <iostream>
6  using std::cout;
7  using std::cin;
8  using std::endl;
9
10 #include <string> // program uses C++ standard string class
11 using std::string;
12 using std::getline;
13
14 // GradeBook class definition
15 class GradeBook
16 {
17 public:
18    // function that sets the course name
19    void setCourseName( string name )
20    {
21       courseName = name; // store the course name in the object
22    } // end function setCourseName
23
24    // function that gets the course name
25    string getCourseName()
26    {
27       return courseName; // return the object's courseName
28    } // end function getCourseName
29
```

set function modifies `private` data

get function accesses `private` data

© 2007 Pearson Ed -All rights reserved.

Public and Private Members (continued)

```
30    // function that displays a welcome message
31    void displayMessage()
32    {
33       // this statement calls getCourseName to get the
34       // name of the course this GradeBook represents
35       cout << "Welcome to the grade book for\n" << getCourseName() << "!"
36          << endl;
37    } // end function displayMessage
38 private:
39    string courseName; // course name for this GradeBook
40 }; // end class GradeBook
41
42 // function main begins program execution
43 int main()
44 {
45    string nameOfCourse; // string of characters to store the course name
46    GradeBook myGradeBook; // create a GradeBook object named myGradeBook
47
48    // display initial value of courseName
49    cout << "Initial course name is: " << myGradeBook.getCourseName()
50       << endl;
51
```

Use *set* and *get* functions, even within the class

private members accessible only to member functions of the class

default constructor

Accessing **private** data outside class definition

© 2007 Pearson Ed -All rights reserved.

Public and Private Members (continued)

```
52    // prompt for, input and set course name
53    cout << "\nPlease enter the course name:" << endl;
54    getline( cin, nameOfCourse ); // read a course name with blanks
55    myGradeBook.setCourseName( nameOfCourse ); // set the course name
56
57    cout << endl; // outputs a blank line
58    myGradeBook.displayMessage(); // display message with new course name
59    return 0; // indicate successful termination
60 } // end main
```

Modifying `private` data outside class definition

```
Initial course name is:

Please enter the course name:
CS101 Introduction to C++ Programming

Welcome to the grade book for
CS101 Introduction to C++ Programming!
```

default setting from constructor is an empty string!!

© 2007 Pearson Ed -All rights reserved.

Software Engineering Observation 19.1

- As a rule of thumb, data members should be declared private
- Member functions should be declared public.
  - Except member functions that are accessed only by other member functions of the class.

- Often useful to have *get* and *set* functions
  - To access private members in controlled ways

© 2007 Pearson Ed -All rights reserved.

Constructors and Destructors

- *Constructor*:– a function used to initialize the data of an object of a class
  - Same name as class itself
  - Cannot return anything, not even **`void`**
  - A class may define more than one *constructor*
    - With different parameter lists
    - Default constructor has no parameters

Compiler provides one if you do not!

- Called automatically
  - When class object is declared as automatic variable
  - By **`new`** operator

Compiler's default simply calls constructors of data members of the class.

Constructors and Destructors (continued)

- *Destructor*:– a function used to clean up an object of a class prior to deleting that object
  - Class name preceeded by `'~'`
  - No parameters, no result

Compiler provides one if you do not!

- Called automatically
  - When function exits scope of automatic class object
  - By **delete**

Compiler's default simply calls destructors of data members of the class.

Constructors and Destructors (continued)

- Constructors – Similar to Java
- Destructors – No counterpart in Java
- Purpose of Destructors
  - Free dynamic storage pointed to only by members of object
  - Reduce reference count when object disappears
  - Safely close things – e.g., files
  - …

Constructor Example

```
1  // Fig. 19.7: fig19_07.cpp
2  // Instantiating multiple objects of the GradeBook class and using
3  // the GradeBook constructor to specify the course name
4  // when each GradeBook object is created.
5  #include <iostream>
6  using std::cout;
7  using std::endl;
8
9  #include <string> // program uses C++ standard string class
10 using std::string;
11
12 // GradeBook class definition
13 class GradeBook
14 {
15 public:
16    // constructor initializes courseName with string supplied as argument
17    GradeBook( string name )
18    {
19       setCourseName( name ); // call set function to initialize courseName
20    } // end GradeBook constructor
21
22    // function to set the course name
23    void setCourseName( string name )
24    {
25       courseName = name; // store the course name in the object
26    } // end function setCourseName
27
```

Constructor has same name as class and no return type

Initialize data member

© 2007 Pearson Ed -All rights reserved.

Constructor Example

```
28    // function to get the course name
29    string getCourseName()
30    {
31       return courseName; // return object's courseName
32    } // end function getCourseName
33
34    // display a welcome message to the GradeBook user
35    void displayMessage()
36    {
37       // call getCourseName to get the cou
38       cout << "Welcome to the grad                ame()
39          << "!" << endl;
40    } // end function displayM
41 private:
42    string courseName; // course name for this GradeBook
43 }; // end class GradeBook
44
```

Destructor need so this class object can free dynamically allocated memory

© 2007 Pearson Ed -All rights reserved.

Constructor Example

```
45 // function main begins program execution
46 int main()
47 {
48    // create two GradeBook objects
49    GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
50    GradeBook gradeBook2( "CS102 Data Structures in C++" );
51
52    // display initial value of courseName for each GradeBook
53    cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
54       << "\ngradeBook2 created for course: " << gradeBook2.getCourseName()
55       << endl;
56    return 0; // indicate successful termination
57 } // end main
```

Creating objects implicitly calls the constructor

```
gradeBook1 created for course: CS101 Introduction to C++ Programming
gradeBook2 created for course: CS102 Data Structures in C++
```

© 2007 Pearson Ed -All rights reserved.

Class in a Separate Header File for Reusability

- **`.cpp`** files for source-code implemenations
  - Class implementations
  - Main programs
  - Test programs
  - **...**
- Header files
  - Separate files in which class definitions are placed.
  - Allow compiler to recognize the classes when used elsewhere.
  - Generally have **`.h`** filename extensions
- Driver file
  - A program used to test software (such as classes).
  - Contains a **`main`** function so it can be executed.

© 2007 Pearson Ed -All rights reserved.

Interfaces *versus* Implementation

- *Interface*
  - Describes what services a class's clients can use and how to request those services.
    - without revealing how the class carries out the services.
    - a class definition listing only *public* member function prototypes.
  - A class's interface consists of the class's public member functions (services).

- Defined in class header file (`.h`)

© 2007 Pearson Ed -All rights reserved.

Interfaces *versus* Implementation

- Implementation of member functions
  - In a separate source-code file for a class
    - Use binary scope resolution operator (::) to tie each member function to the class definition.
  - Implementation details are hidden.
    - Client code does not need to know the implementation.

© 2007 Pearson Ed -All rights reserved.

Interfaces *versus* Implementation (Example)

```
1  // Fig. 19.11: GradeBook.h
2  // GradeBook class definition. This file presents GradeBook's public
3  // interface without revealing the implementations of GradeBook's member
4  // functions, which are defined in GradeBook.cpp.
5  #include <string> // class GradeBook uses C++ standard string class
6  using std::string;
7
8  // GradeBook class definition
9  class GradeBook
10 {
11 public:
12    GradeBook( string ); // constructor that initializes courseName
13    void setCourseName( string ); // function that sets the course name
14    string getCourseName(); // function that gets the course name
15    void displayMessage(); // function that displays a welcome message
16 private:
17    string courseName; // course name for this GradeBook
18 }; // end class GradeBook
```

© 2007 Pearson Ed -All rights reserved.

Interfaces *versus* Implementation (continued)

```
1  // Fig. 19.12: GradeBook.cpp
2  // GradeBook member-function definitions. This file contains
3  // implementations of the member functions prototyped in GradeBook.h.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  #include "GradeBook.h" // include definition of class GradeBook
9
10 // constructor initializes courseName with string supplied as argument
11 GradeBook::GradeBook( string name )
12 {
13    setCourseName( name ); // call set function to initialize courseName
14 } // end GradeBook constructor
15
16 // function to set the course name
17 void GradeBook::setCourseName( string name )
18 {
19    courseName = name; // store the course name in the object
20 } // end function setCourseName
21
```

`GradeBook` implementation is placed in a separate source-code file

Include the header file to access the class name `GradeBook`

Binary scope resolution operator ties a function to its class

© 2007 Pearson Ed -All rights reserved.

Interfaces *versus* Implementation (continued)

```
22 // function to get the course name
23 string GradeBook::getCourseName()
24 {
25    return courseName; // return object's courseName
26 } // end function getCourseName
27
28 // display a welcome message to the GradeBook user
29 void GradeBook::displayMessage()
30 {
31    // call getCourseName to get the courseName
32    cout << "Welcome to the grade book for\n" << getCourseName()
33       << "!" << endl;
34 } // end function displayMessage
```

© 2007 Pearson Ed -All rights reserved.

Client of the Interface

```
1  // Fig. 19.13: fig19_13.cpp
2  // GradeBook class demonstration after separating
3  // its interface from its implementation.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  #include "GradeBook.h" // include definition of class GradeBook
9
10 // function main begins program execution
11 int main()
12 {
13    // create two GradeBook objects
14    GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
15    GradeBook gradeBook2( "CS102 Data Structures in C++" );
16
17    // display initial value of courseName for each GradeBook
18    cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
19       << "\ngradeBook2 created for course: " << gradeBook2.getCourseName()
20       << endl;
21    return 0; // indicate successful termination
22 } // end main
```

```
gradeBook1 created for course: CS101 Introduction to C++ Programming
gradeBook2 created for course: CS102 Data Structures in C++
```

© 2007 Pearson Ed -All rights reserved.

Summary

- Introduced class definitions and objects
  - Public versus private access into class.
- Syntax for member functions
- Syntax data members
  - Get and Set functions
  - Constructors & Destructors
- Placing classes in separate files
- Separating interface from implementation

Questions?

Reading:–
Deitel & Deitel, Chapter 19