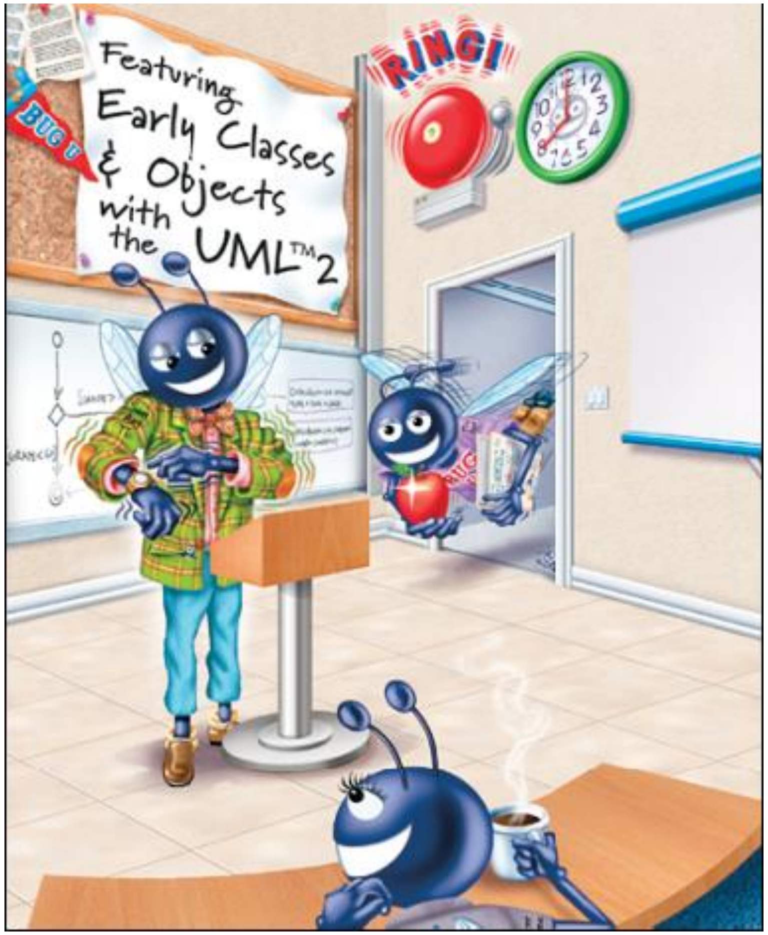
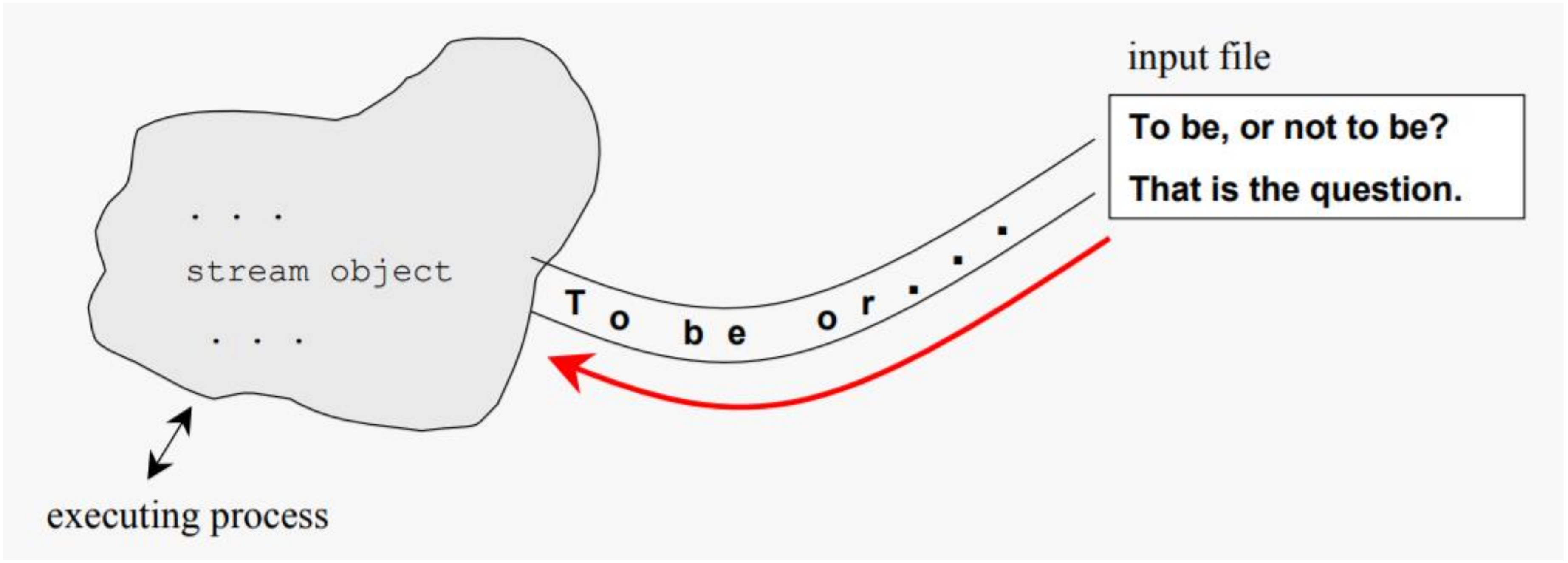
C++程序设计



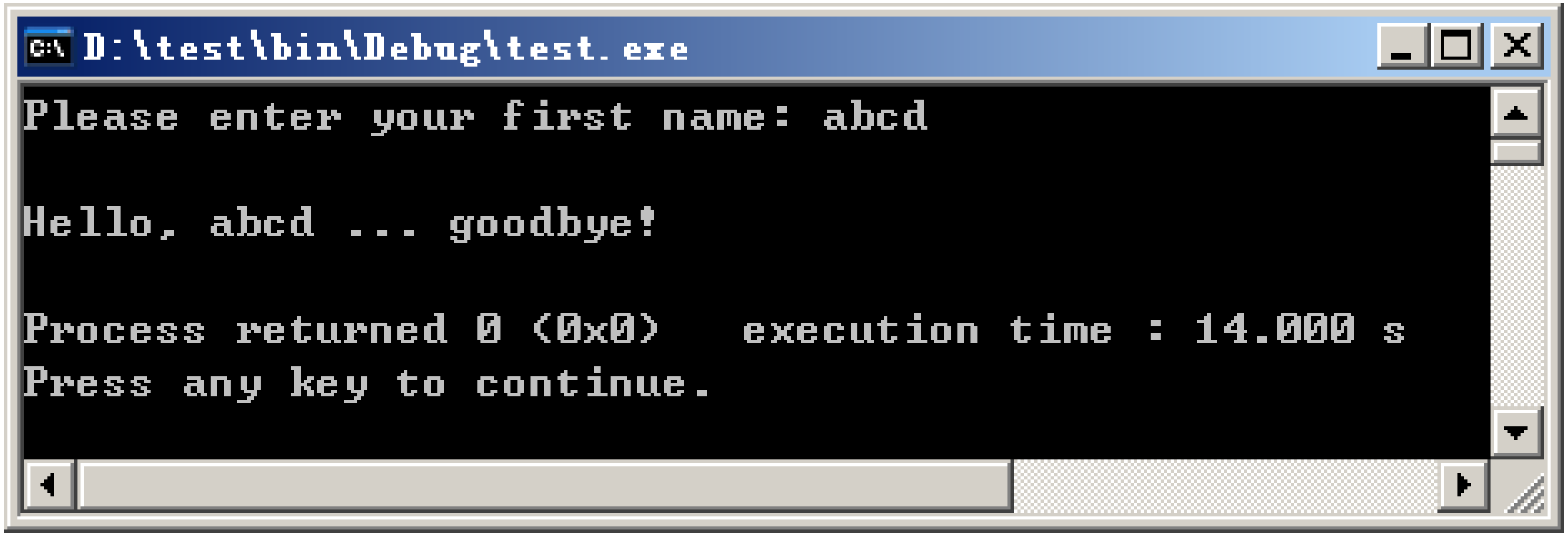
上节课内容回顾

1. C++中的输入输出语句
2. 算术运算符；关系运算符；位运算符

上节课内容回顾


input file
To be, or not to be?
That is the question.
stream object
. . .
. . .
T o b e o r . . .
executing process

上节课内容回顾



```
Please enter your first name: abcd

Hello, abcd ... goodbye!

Process returned 0 (0x0)   execution time : 14.000 s
Press any key to continue.
```

上节课内容回顾

```
#include <iostream>
using namespace std;
int main()
{
   char name[10];
   cout << "Please enter your first name: ";
   cin >> name;
   cout << endl;
   cout << "Hello, " << name;
   cout << "... goodbye!" << endl;
   return 0;
}
```

上节课内容回顾

```
#include <iostream>
using namespace std;
int main()
{
   string name;
   cout << "Please enter your first name: ";
   cin >> name;
   cout << endl;
   cout << "Hello, " << name;
   cout << "... goodbye!" << endl;
   return 0;
}
```

cin.get(char* s, streamsize n, char delim)

```
int main()
{
   char ch, a[20];
   cout << "Input some characters: ";
   cin.get(a,5,'d');
   cout << endl << a << endl;
   cout << cin.gcount() << endl;
   //cin.get(ch);
   //cout << (int)ch;
   return 0;
}
```

cin.getline (char* s, streamsize n);

```
int main () {
   char ch,name[256], title[256];

   cout << "Please, enter your name:  ";
   cin.getline (name,256);

   cout << "Please, enter your favourite movie:";
   cin.getline (title,256);

   cout << name << "'s favourite movie is " << title;

   return 0;
}
```

cin.fail, clear, ignore

```
int main() {
    int x;
    cin >> x;
    while(cin.fail()) {
        cout << "Error" << endl;
        cin.clear();
        cin.ignore(256,'\n');
        cin >> x;
    }
    cout << x << endl;

    return 0;
}
```

上节课内容回顾

思考题：

要求输入一个5位整数，分解出它的每位数字，每个数字间隔3个空格进行打印。

上节课内容回顾

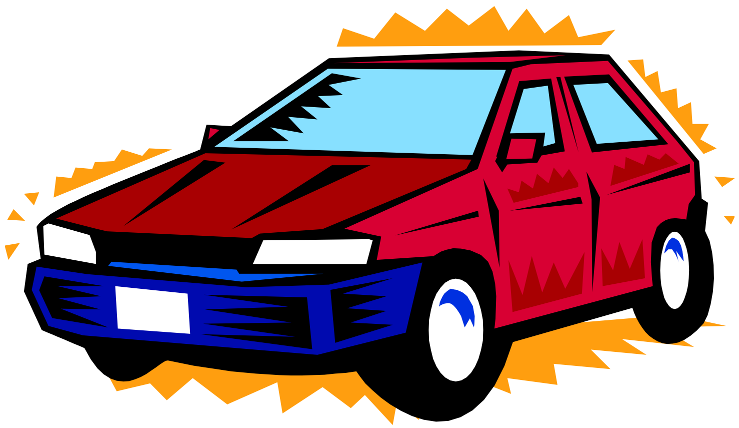
```
…
int number; // integer read from user

cout << "Enter a five-digit integer: "; // prompt
cin >> number; // read integer from user

cout << number / 10000 << "   ";
number = number % 10000;
cout << number / 1000 << "   ";
number = number % 1000;
…
```

第三讲 类和对象介绍

学习目标：

- 如何定义类
- 如何调用成员函数
- 构造函数
- 实现与接口分离

1. 类，对象，成员函数 和 数据成员



车型：法拉利

颜色：红色

年份：1995

活动

发动

停车

加速

2. 本章中的示例预览

- GradeBook 类
- 涉及到:
  - 成员函数（Member functions）
  - 数据成员（Data members）
  - 类的客户（Clients of a class）
  - 接口与实现分离（Separating interface from implementation）
  - 数据验证（Data validation）

3. 定义带有一个成员函数的类

- 类的定义
  - 通知编译器哪些数据成员和成员函数属于该类
  - 关键字 class
  - 定义体在花括号内 ({ })
    - 声明数据成员和成员函数
    - 访问修饰符 public:
      - 其他函数和其他类的成员函数可以访问

3. 定义带有一个成员函数的类

```
// GradeBook class definition
class GradeBook
{
public:
   // function that displays a welcome message to the GradeBook user
   void displayMessage()
   {
      cout << "Welcome to the Grade Book!" << endl;
   } // end function displayMessage
}; // end class GradeBook
```

Beginning of class definition for class **GradeBook**

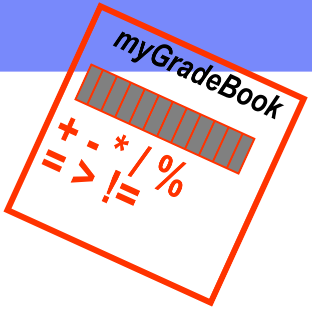
Beginning of class body

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

Member function **displayMessge** returns nothing

End of class body

3. 定义带有一个成员函数的类



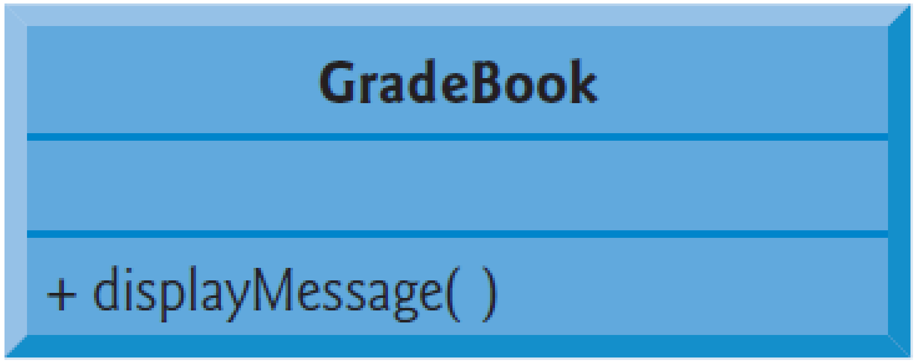
```
// function main begins program execution
int main()
{
   GradeBook myGradeBook; // crea
   myGradeBook.displayMessage(); // call object's displayMessage function
   return 0; // indicate successful termination
} // end main
```

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

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

3. 定义带有一个成员函数的类

GradeBook类的UML(Unified Modeling Language)类图



UML class diagram indicating that class GradeBook has a public displayMessage operation.

3. 定义带有一个成员函数的类

- UML 类图
  - 由三部分组成的矩形
    - 顶部包含水平居中、黑体的类的名字
    - 中部包含类的属性
    - 底部包含类的操作
      - 操作前面的 (+) 表示该操作为 public

4. 定义带有一个参数的成员函数

- 函数参数
  - 函数需要客户提供相关信息来完成任务
  - 客户在函数调用时所提供的参数值拷贝给函数的参数

4. 定义带有一个参数的成员函数

- A string
  - 字符集合
  - C++ 标准类库 std::string
    - 需包含 <string>
- getline 函数
  - 读取一行输入
  - 例：
    - getline(cin, nameOfCourse);

4. 定义带有一个参数的成员函数

```
#include <string> // program uses C++ standard string class
...
class GradeBook
{
   public:
     // function that displays a welcome message to the GradeBook user
     void displayMessage( string courseName )
     {
       cout << "Welcome to the grade book for\n" << courseName << "!"
          << endl;
     } // end function displayMessage
}; // end class GradeBook
```

Include string class definition

Member function parameter

Use the function parameter as a variable

4. 定义带有一个参数的成员函数

```
int main()
{
   string nameOfCourse;
   GradeBook myGradeBook;
   // prompt for and input course name
   cout << "Please enter the course name:" << endl;
   getline( cin, nameOfCourse ); // read a course name with blanks
   cout << endl; // output a blank line
   myGradeBook.displayMessage( nameOfCourse );
   return 0; // indicate successful termination
} // end main
```

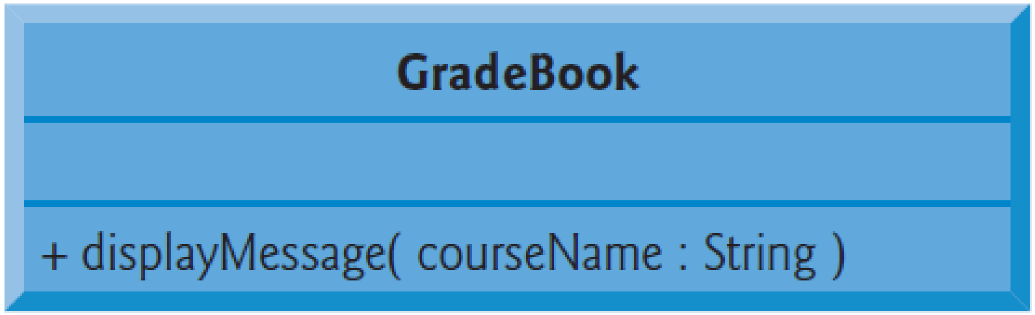
Passing an argument to the member function

4. 定义带有一个参数的成员函数

- 参数列表
  - UML 中的表示方式
    - 在成员函数的“( )”中，参数名称 : 参数类型

4. 定义带有一个参数的成员函数

GradeBook类的UML类图



UML class diagram indicating that class GradeBook has a displayMessage operation with a courseName parameter of UML type String.

5. 数据成员，get 和 set 函数

- 局部变量
  - 在函数体定义内部声明的变量
    - 不能在函数体外部使用
  - 当函数终止
    - 局部变量将被销毁
- 属性
  - 存在于对象的整个生命周期内
  - 表示为数据成员
    - 即类定义中的变量
  - 每个对象维护一份自己的属性拷贝

```
class GradeBook
{
public:
   // function that sets the course name
   void setCourseName( string name )
   {
      courseName = name; // store the course name in the object
   } // end function setCourseName

   // function that gets the course name
   string getCourseName()
   {
      return courseName; // return the object's courseName
   } // end function getCourseName
```

set function modifies **private** data

get function accesses **private** data

```
    // function that displays a welcome message
    void displayMessage()
    {
      // this statement calls getCourseName to get the
      // name of the course this GradeBook represents
      cout << "Welcome to the grade book for\n" << getCourseName() << "!"
        << endl;
    } // end function displayMessage
  private:
    string courseName; // course name for this GradeBoo
  }; // end class G
```

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

private members accessible only to member functions of the class

```
int main()
 {
   string nameOfCourse; // string of characters to store the course name
   GradeBook myGradeBook; // create a GradeBook object named myGradeBook
   // display initial value of courseName

   cout << "Initial course name is: " << myGradeBook.getCourseName() << endl;

   // prompt for, input and set course name
   cout << "\nPlease enter the course name
   getline( cin, nameOfCourse ); // read a co
   myGradeBook.setCourseName( nameOfCourse ); // set the course name


   cout << endl; // outputs a blank line
   myGradeBook.displayMessage(); // display message with new course name
   return 0; // indicate successful termination
} // end main
```

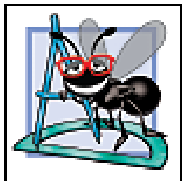
Accessing **private** data outside class definition

Modifying **private** data outside class definition

5. 数据成员， get 和 set 函数

- 访问修饰符 private
  - 使得数据成员或成员函数只能由类的成员函数访问
  - 类成员的默认访问为 private
  - 数据隐藏

5. 数据成员，get 和 set 函数



软件工程知识：根据经验，数据成员应该声明为private，成员函数应该声明为public，如果某些成员函数只是被该类的其他成员函数访问，那么它们更适合声明为private。

5. 数据成员，get 和 set 函数

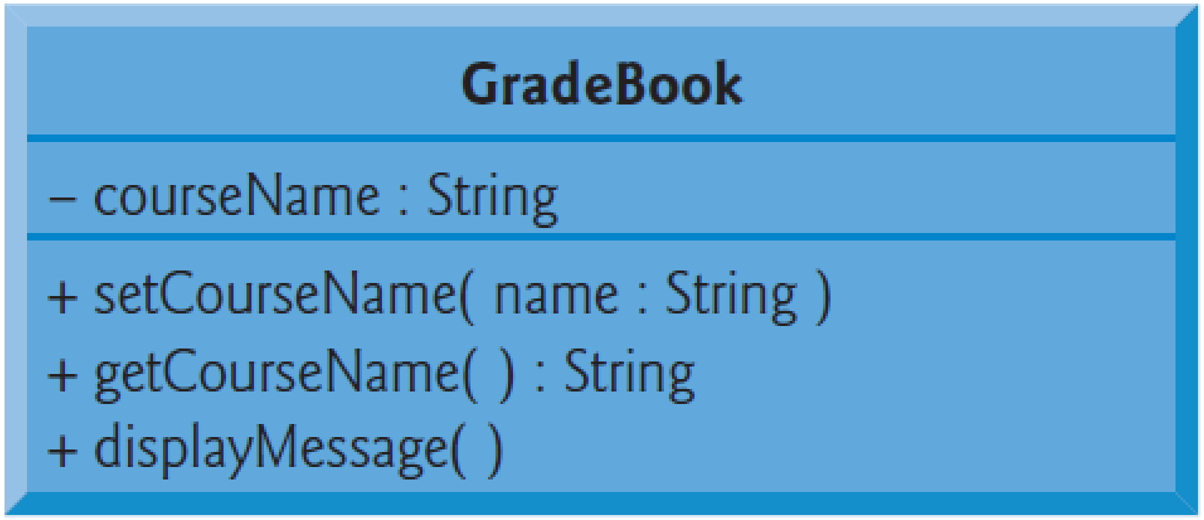
- Software engineering with *set* and *get* functions
  - public 成员函数允许类的客户 set 和 get 类的 private 数据成员
  - *set* 函数有时被称为 mutator (更换器)，*get* 有时被称为 accessor (访问器)
  - 允许类的创建者来控制客户如何来访问 private 数据

5. 数据成员，get 和 set 函数

- UML diagram
  - 标识操作的返回值类型
    - 在函数名的 “( )” 后加 “ : ” 和返回值类型
  - 数据成员名称前面的 “ - ” 标识该成员为 private 成员

5. 数据成员， get 和 set 函数

GradeBook类的UML类图



UML class diagram for class GradeBook with a private courseName attribute and public operations setCourseName, getCourseName and displayMessage.

6. 利用构造函数来初始化对象

- Constructors (构造函数)
  - 用来在对象创建时来初始化对象数据的函数
    - 当对象创建时被隐式调用
    - 必须与类同名
    - 不能有返回值
  - 缺省的构造函数没有参数
    - 当类没有定义构造函数时，编译器会提供缺省的构造函数

```
// GradeBook class definition
class GradeBook
{
public:
   // constructor initializes courseName with string supplied as argument
   GradeBook( string name )
   {
      setCourseName( name ); // call set function to initialize courseName
   } // end GradeBook constructor

......
```

Constructor has same name as class and no return type

Initialize data member

```
int main()
 {
   // create two GradeBook objects
   GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
   GradeBook gradeBook2( "CS102 Data Structures in C++" );


   // display initial value of courseName for each GradeBook
  cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
     << "\ngradeBook2 created for course: " << gradeBook2.getCourseName()
     << endl;
  return 0; // indicate successful termination
} // end main
```

Creating objects implicitly calls the constructor

6. 利用构造函数来初始化对象

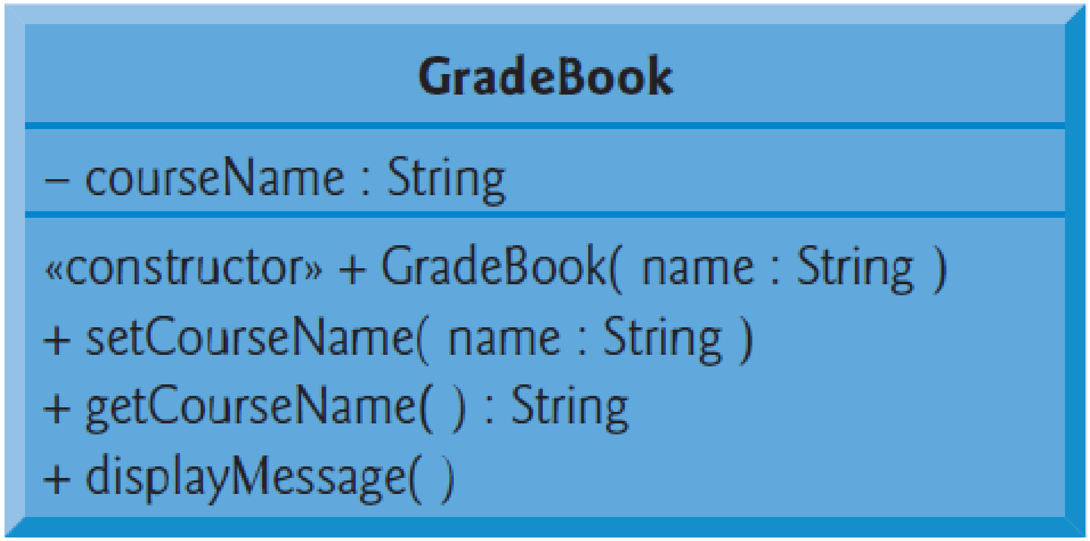
错误预防技巧：除非没有必要初始化类的数据成员，否则请提供构造函数，这样可以保证当类的每个新对象被创建时，类的数据成员都用有意义的值进行了初始化。

6. 利用构造函数来初始化对象

- Constructors in a UML class diagram
  - 在操作部分出现
  - 为了与其他操作进行区分
    - 在构造函数名前加：<<constructor>>
  - 通常放置在其他操作之前

6. 利用构造函数来初始化对象

GradeBook类的UML类图


GradeBook
– courseName : String
«constructor» + GradeBook(name : String)
+ setCourseName(name : String)
+ getCourseName( ) : String
+ displayMessage( )

UML class diagram indicating that class GradeBook has a constructor with a name parameter of UML type String.

7. 将类放到单独的文件中来提高重用性

- .cpp： 源文件
- Header files （.h）
  - 放置类的定义
    - 允许编译器在其他地方识别该类
- Driver files
  - 用来测试软件的程序 (如：测试 classes)
  - 包含 main 函数，可以被执行

```
// GradeBook.h
// GradeBook class definition in a separate file from main.
……
// GradeBook class definition
class GradeBook
{
public:
  // constructor initializes courseName with string supplied as argument
  GradeBook( string name )
  {
    setCourseName( name ); // call set function to initialize courseName
  } // end GradeBook constructor
……
};
```

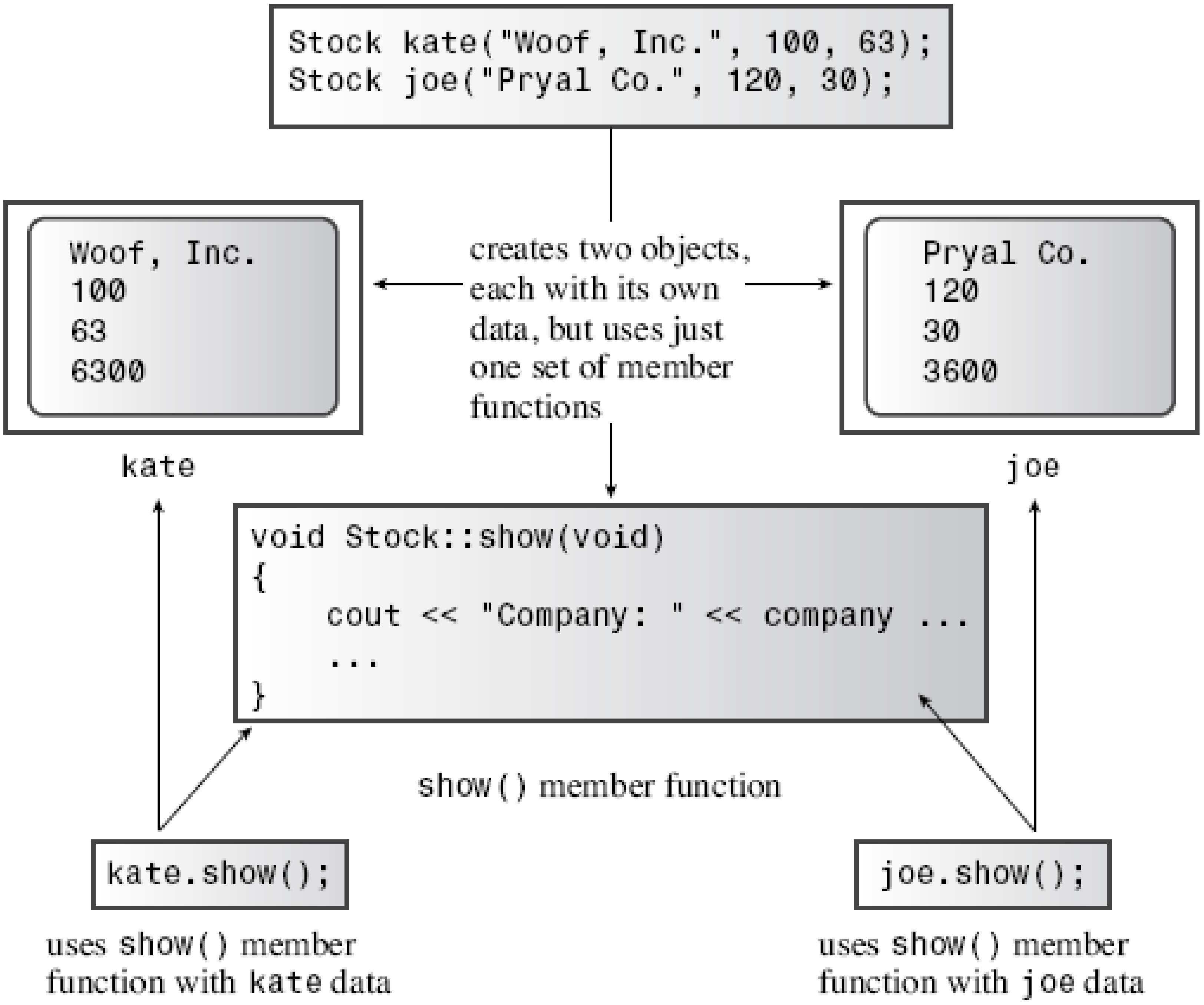
Class definition is in a header file

```
......
#include "GradeBook.h" // include definition of class GradeBook


// function main begins program execution
int main()
{
   // create two GradeBook objects
   GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
   GradeBook gradeBook2( "CS102 Data Structures in C++" );

......
}
```

Including the header file causes the class definition to be copied into the file

7. 将类放到单独的文件中来提高重用性

- **Creating objects**
  - 编译器必须知道对象的大小
    - 对象的大小为类的数据成员的大小
    - 编译器创建一份类的成员函数的拷贝，该拷贝为所有类的对象所共享



```
Stock kate("Woof, Inc.", 100, 63);
Stock joe("Pryal Co.", 120, 30);
```

```
void Stock::show(void)
{
    cout << "Company: " << company ...
    ...
}
```

```
kate.show();
```

```
joe.show();
```

Woof, Inc.
100
63
6300
kate
creates two objects, each with its own data, but uses just one set of member functions
Pryal Co.
120
30
3600
joe
show() member function
uses show() member function with kate data
uses show() member function with joe data

8. 接口与实现分离

- Interface (接口)
  - 描述客户能够使用类的哪些服务，如何请求这些服务
    - 包含成员函数名称、返回类型、参数类型的类定义，即：函数原型
  - 类的接口包含类的public成员函数 (services)

8. 接口与实现分离

- **Separating interface from implementation**
  - 如果服务的实现改变，只要接口保持不变，客户端的代码就无需改变
  - 在另外一个源文件内来实现类的成员函数
  - 在该源文件中用二元解析运算符（::）将成员函数与类联系起来
  - 客户代码无需知晓实现细节
  - 在头文件中声明该类提供的接口

```
//// Fig. 3.11: GradeBook.h
……
// GradeBook class definition
class GradeBook
{
public:
  GradeBook( string ); // constructor that initializes courseName
  void setCourseName( string ); // function that sets the course name
  string getCourseName(); // function that gets the course name
  void displayMessage(); // function that displays a welcome message
private:
  string courseName; // course name for this GradeBook
}; // end class GradeBook
```

Interface contains data members and member function prototypes

```
// GradeBook.cpp
......
#include "GradeBook.h" // include definition of class GradeBook


// constructor initializes courseName with string supplied as argument
GradeBook::GradeBook( string name
 {
   setCourseName( name ); // call set function to initialize courseName
} // end GradeBook constructor


// function to set the course name
void GradeBook::setCourseName( string name )
 {
   courseName = name; // store the course name in the object
} // end function setCourseName
```

GradeBook implementation is placed in a separate source-code file

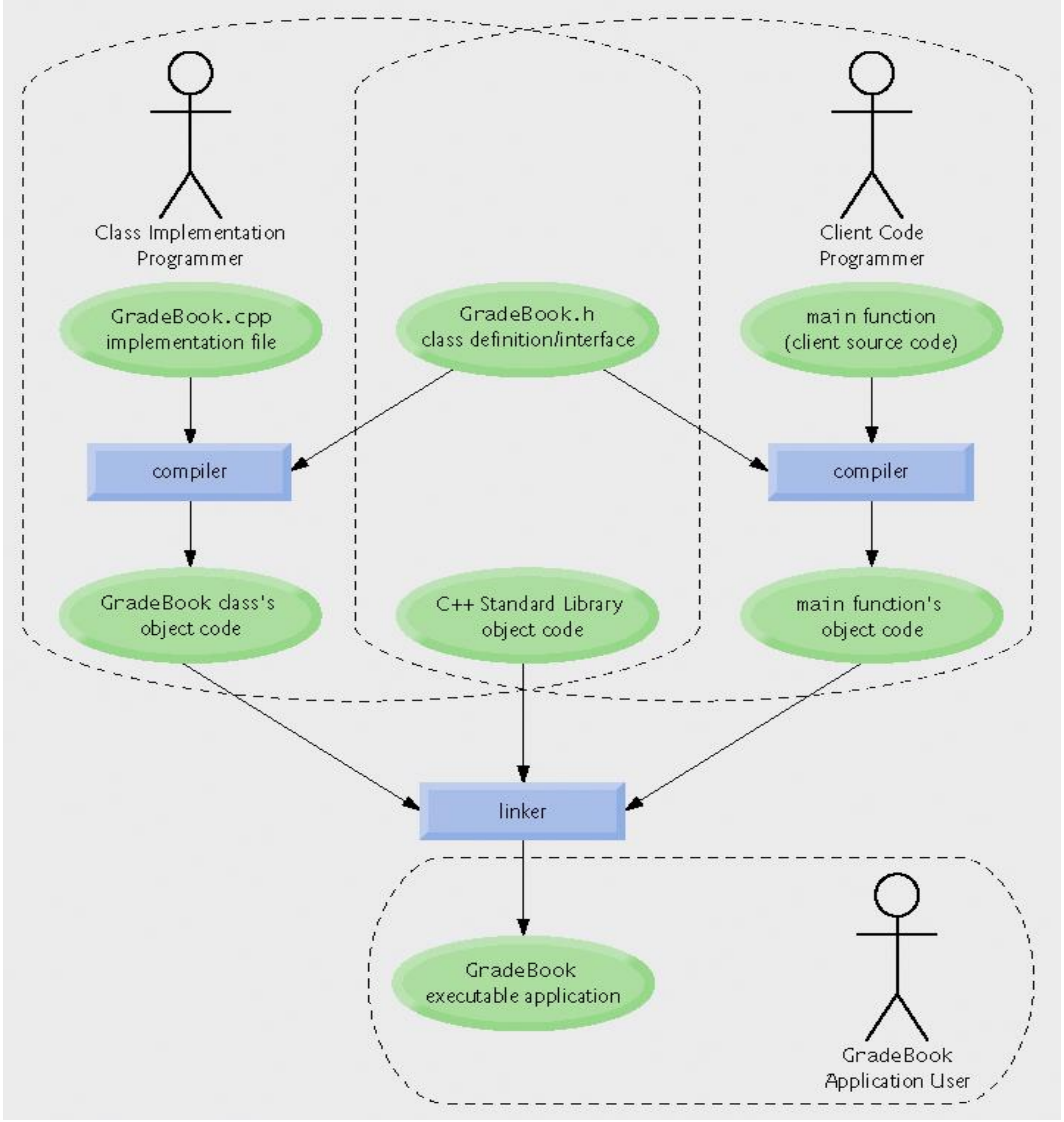
Include the header file to access the class name **GradeBook**

Binary scope resolution operator ties a function to its class

```
#include <iostream>

……
#include "GradeBook.h" // include definition of class GradeBook
// function main begins program execution
int main()
 {
   GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
   GradeBook gradeBook2( "CS102 Data Structures in C++" );
   cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
     << "\ngradeBook2 created for course: " << gradeBook2.getCourseName()
     << endl;
   return 0; // indicate successful termination
} // end main
```

8. 接口与实现分离

- **The Compilation and Linking Process**
  - 源文件被编译为类的目标代码
    - 只需向客户提供头文件和目标代码
  - 客户必须在自己的代码中包含头文件
  - 创建可执行程序
    - 客户程序的目标代码必须与类的目标代码和在应用程序中用到的 C++ 标准类库的目标代码进行连接


Class Implementation Programmer
Client Code Programmer
GradeBook.cpp implementation file
GradeBook.h class definition/interface
main function (client source code)
compiler
compiler
GradeBook class's object code
C++ Standard Library object code
main function's object code
linker
GradeBook executable application
GradeBook Application User

9. 利用 set 函数进行数据验证

- *set* functions can validate (验证) data
  - ➢ 有效性检查
  - ➢ 保持对象的数据成员具有有效值
  - ➢ 可以通过返回值来指示设置无效数据
- string member functions
  - ➢ length – 返回字符串的长度
  - ➢ substr – 返回字符串的子串

```
#include <iostream>
using std::cout;
using std::endl;

#include "GradeBook.h" // include definition of class GradeBook

// constructor initializes courseName with string supplied as argument
GradeBook::GradeBook( string name )
{
   setCourseName( name ); // validate and store courseName
} // end GradeBook constructor
```

Constructor calls *set* function to perform validity checking

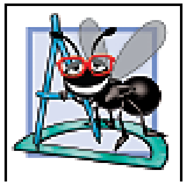
```
void GradeBook::setCourseName( string name )
{
   if ( name.length() <= 25 ) // if name has 25 or fewer characters
     courseName = name; // store the course n

  if ( name.length() > 25 ) // if name has more than 25 characters
  {
     // set courseName to first 25 characters of parameter name
    courseName = name.substr( 0, 25 ); // start at 0, length of 25


    cout << "Name \"" << name << "\" exceeds maximum length (25).\n"
         << "Limiting courseName to first 25 characters.\n" << endl;
   } // end if
} // end function setCourseName
```

set functions perform validity checking to keep **courseName** in a consistent state

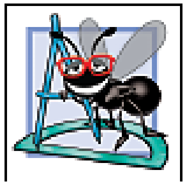
```
int main()
 {
   GradeBook gradeBook1( "CS101 Introduction to Programming in C++" );
   GradeBook gradeBook2( "CS102 C++ Data Structures" );


   // display each GradeBook's courseName
   cout << "gradeBook1's initial course name is: " << gradeBook1.getCourseName()
    << "\ngradeBook2's initial course name is: " << gradeBook2.getCourseName() << endl;

   // modify myGradeBook's courseName (with a valid-length string)
   gradeBook1.setCourseName( "CS101 C++ Programming" );


   // display each GradeBook's courseName
   cout << "\ngradeBook1's course nar
    << "\ngradeBook2's course name is: " << gradeBook2.getCourseName() << endl;
   return 0; // indicate successful termination
 } // end main
```

Constructor will call *set* function to perform validity checking

Call *set* function to perform validity checking



软件工程知识：把数据成员设置成private，而由public成员函数控制访问数据成员的权力，尤其是写的权力，将有助于保证数据的完整性。



软件工程知识：设置private数据值的成员函数应当核实所设置的新值是否正确，如果不正确，设置函数应该将private数据成员置于适当的状态中。

思考题：

- P89 3.11 扩充GradeBook类
  - 包括第二个string数据成员，它表示授课教师的名字
  - 提供一个可以改变教师姓名的设置函数，以及一个可以得到该名字的获取函数
  - 修改构造函数，制定两个参数，一个针对课程名称，另一个针对教师姓名
  - 修改成员函数displayMessage，使得它首先输出欢迎信息和课程名称，然后输出“This course is presented by:”，后跟教师姓名

思考题：

- P89 3.12 Account类
  - 包括一个类型为 int 的数据成员，表示账户余额
  - 提供一个构造函数，接收初始余额并用它初始化数据成员
  - 构造函数应确认初始余额的有效性，保证其大于等于0，否则，余额应设置为0，并显示一条错误信息，指示初始余额无效
  - 成员函数credit将一笔金额加到当前余额中
  - 成员函数debit从Account中取钱，并保证取出金额不超过此Account的余额，否则，余额不变，打印一条信息： “Debit amount exceeded account balance.”
  - 成员函数getBalance返回当前余额
  - 编写一个测试程序，创建两个Account对象，测试该类的成员函数