Object Oriented Programming

Shaobai Kan

chapter 9

Outline

Outline

- Placing a class in a separate file for reusability

Outline

- Placing a class in a separate file for reusability
- Separating interface from implementation

Outline

- Placing a class in a separate file for reusability
- Separating interface from implementation
- Exercise

Placing a class in a separate file for reusability

Separating file for reusability

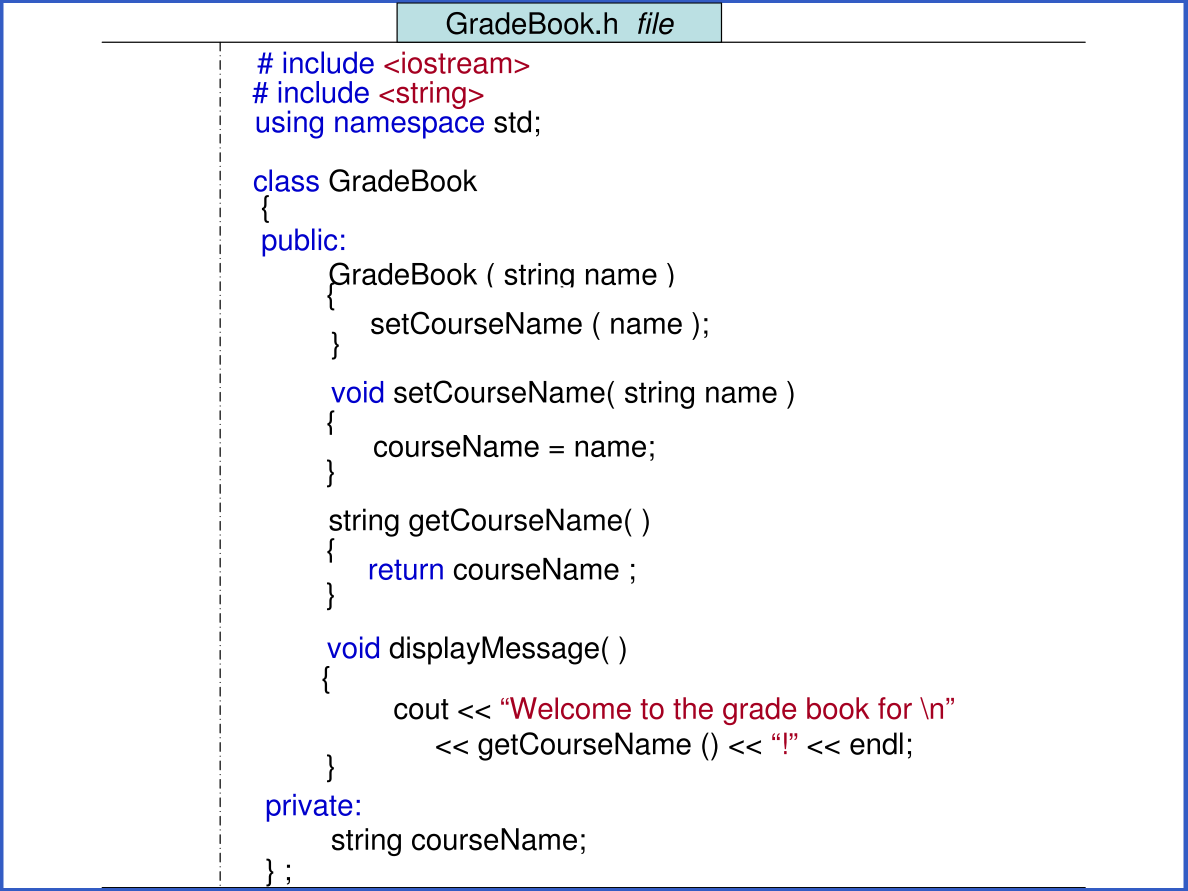
Fact. One of the benefits of creating class definitions is that, *when packaged properly*, our classes can be reused by programmers - potentially worldwide.

Separating file for reusability

Fact. One of the benefits of creating class definitions is that, *when packaged properly*, our classes can be reused by programmers - potentially worldwide.

Example. We can reuse C++ Standard Library type string in any C++ program just by including the header file <string>.

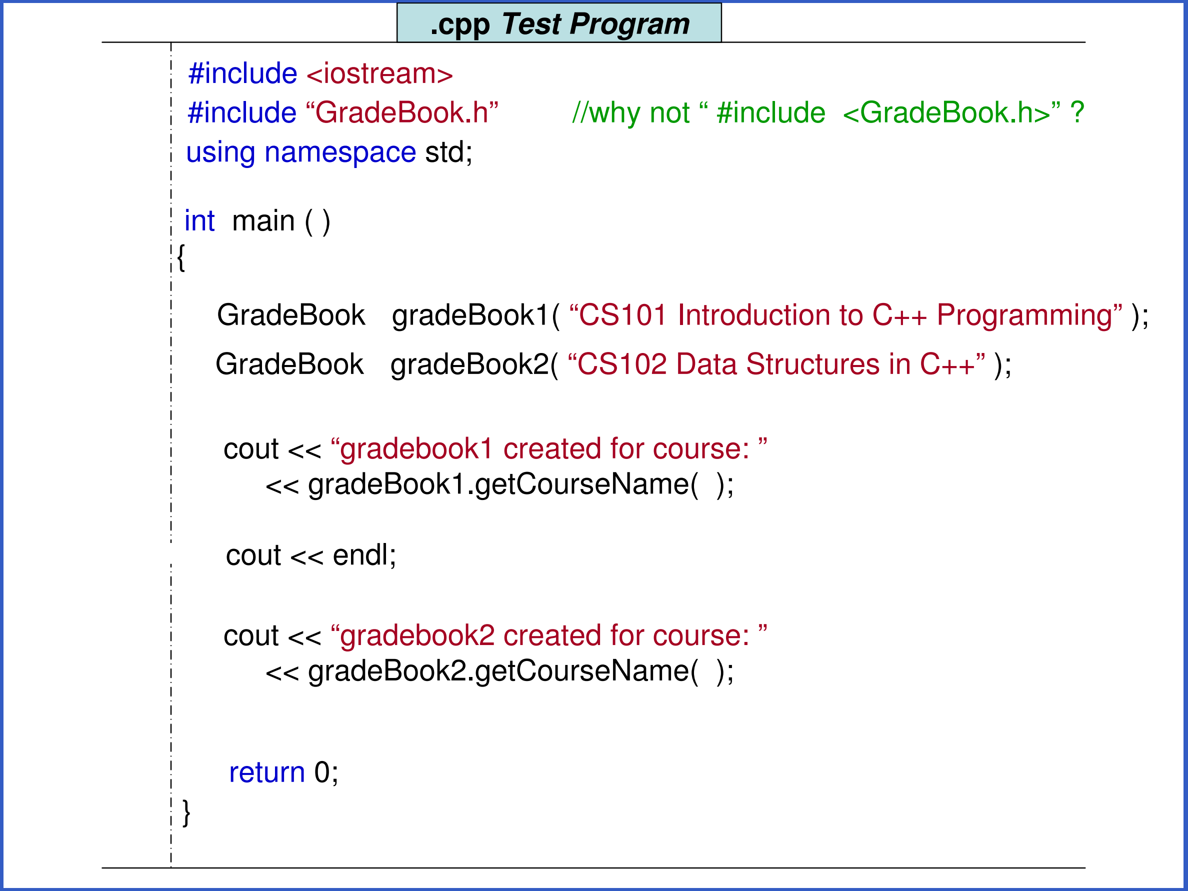
Example: class GradeBook II



GradeBook.h *file*

```
# include <iostream>
# include <string>
using namespace std;

class GradeBook
{
public:
        GradeBook ( string name )
        {
            setCourseName ( name );
        }

        void setCourseName( string name )
        {
            courseName = name;
        }

        string getCourseName( )
        {
            return courseName ;
        }

        void displayMessage( )
        {
                cout << "Welcome to the grade book for \n"
                     << getCourseName () << "!" << endl;
        }
private:
        string courseName;
} ;
```

Example: class GradeBook II



.cpp *Test Program*

```
#include <iostream>
#include "GradeBook.h"       //why not " #include  <GradeBook.h>" ?
using namespace std;

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

     cout << "gradebook1 created for course: "
         << gradeBook1.getCourseName(  );

     cout << endl;

     cout << "gradebook2 created for course: "
         << gradeBook2.getCourseName(  );


     return 0;
}
```

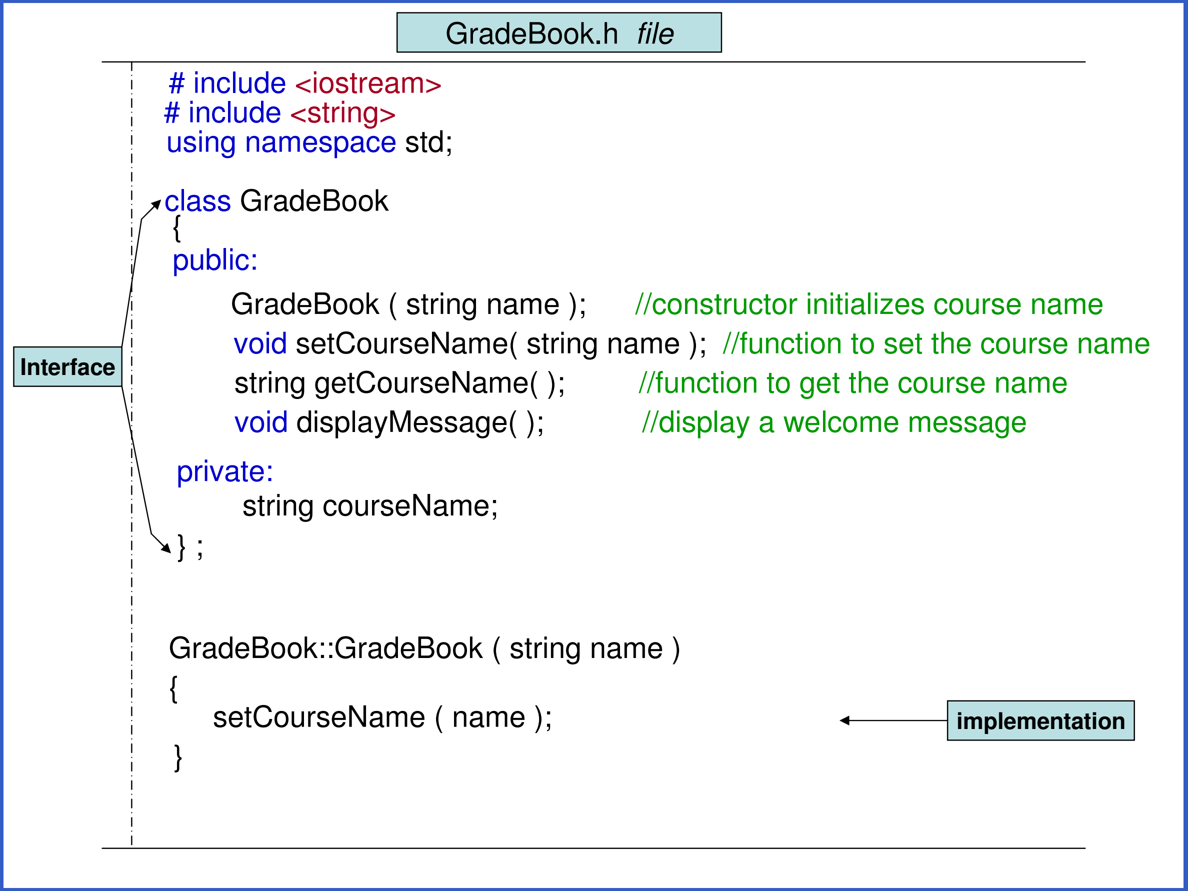
How header files are located?

- When the preprocessor encounters a header file name in quotes (" "), it attempts to locate the header file in the same directory as the file in which the #include directive appears. If the preprocessor cannot find the header file in that directory, it searches for it in the same location(s) as the C++ Standard Library header files.

How header files are located?

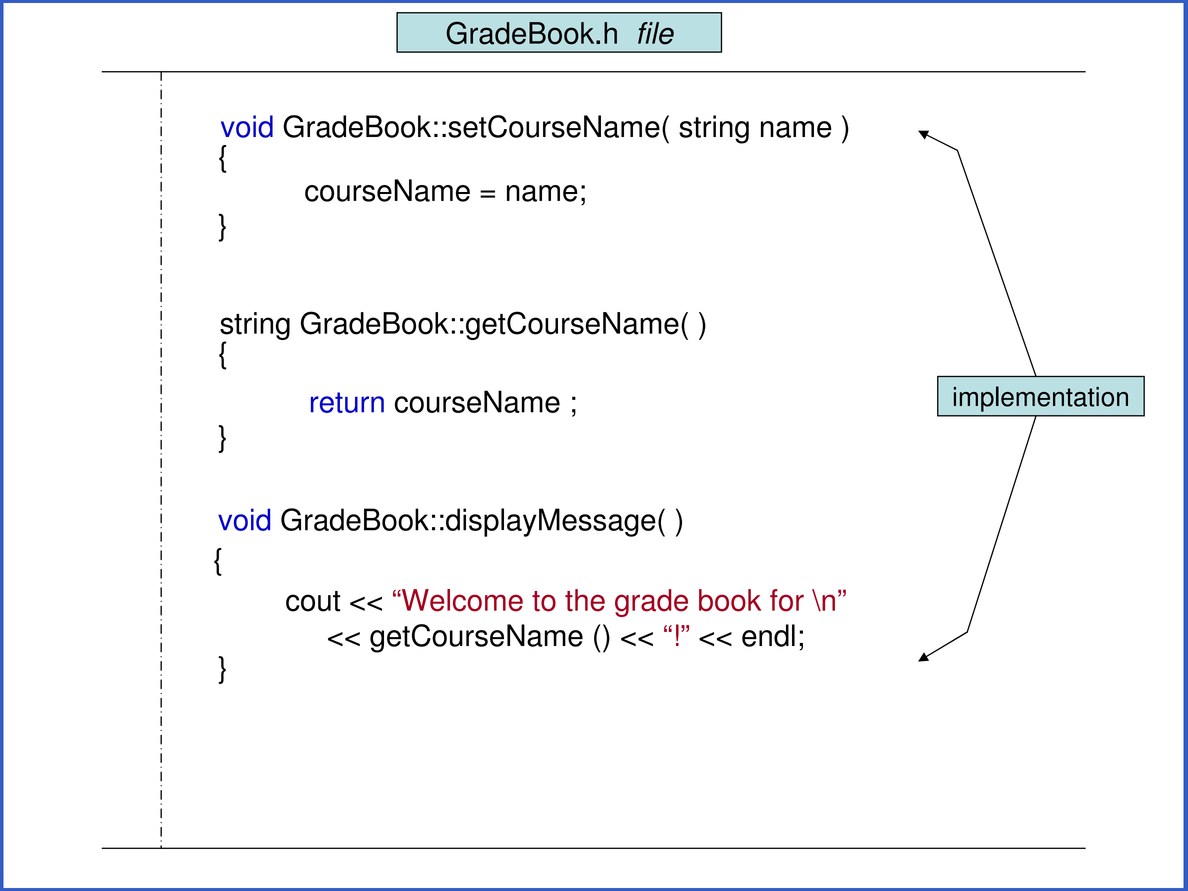
- When the preprocessor encounters a header file name in quotes (" "), it attempts to locate the header file in the same directory as the file in which the #include directive appears. If the preprocessor cannot find the header file in that directory, it searches for it in the same location(s) as the C++ Standard Library header files.
- When the preprocessor encounters a header file name in angle brackets(< >), it assumes that the header file is part of the C++ Standard Library and does not look in the directory of the program that is being processed.

Example: class GradeBook III



```
 # include <iostream>
 # include <string>
 using namespace std;

 class GradeBook
  {
  public:
        GradeBook ( string name );      //constructor initializes course name
        void setCourseName( string name );  //function to set the course name
        string getCourseName( );        //function to get the course name
        void displayMessage( );         //display a welcome message

  private:
         string courseName;
 } ;


 GradeBook::GradeBook ( string name )
 {
     setCourseName ( name );
 }
```

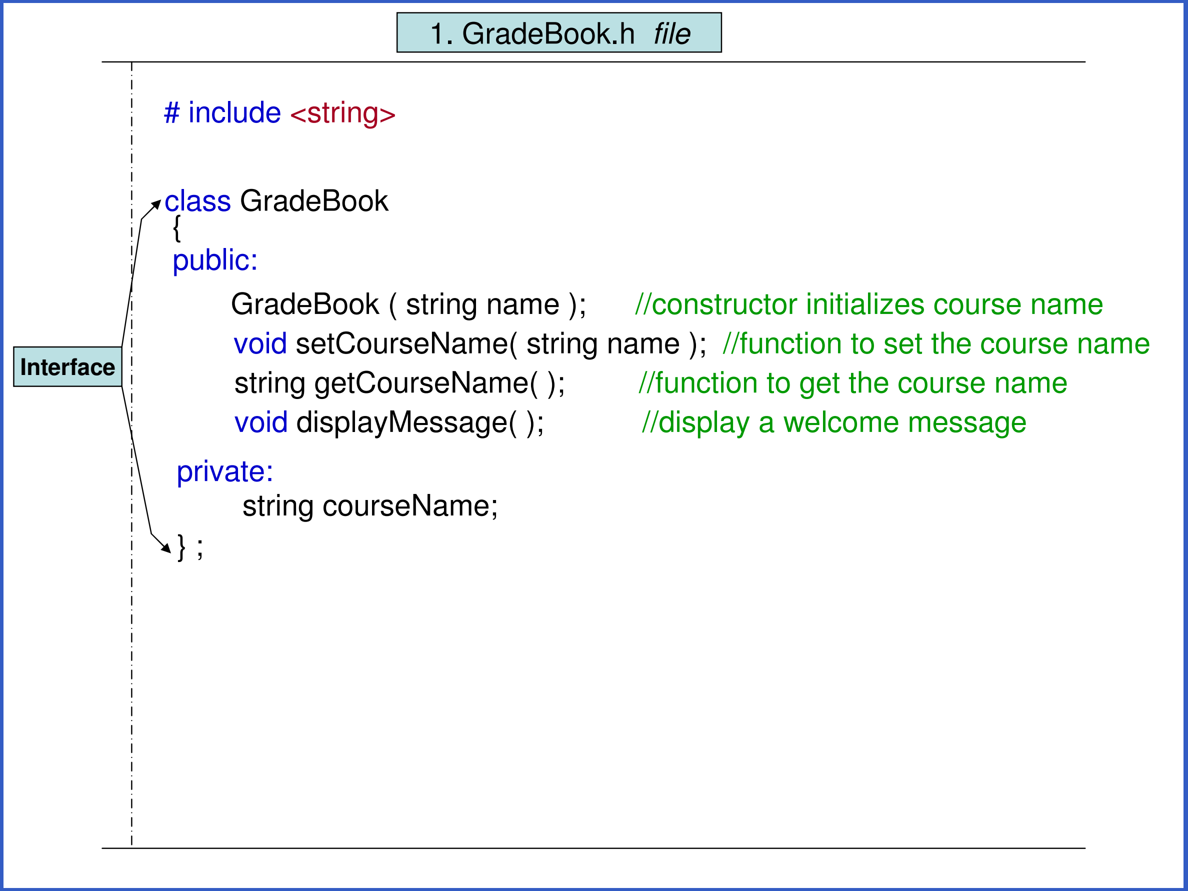
Example: class GradeBook III



```
void GradeBook::setCourseName( string name )
{
        courseName = name;
}


string GradeBook::getCourseName( )
{
        return courseName ;
}

void GradeBook::displayMessage( )
{
        cout << "Welcome to the grade book for \n"
            << getCourseName () << "!" << endl;
}
```

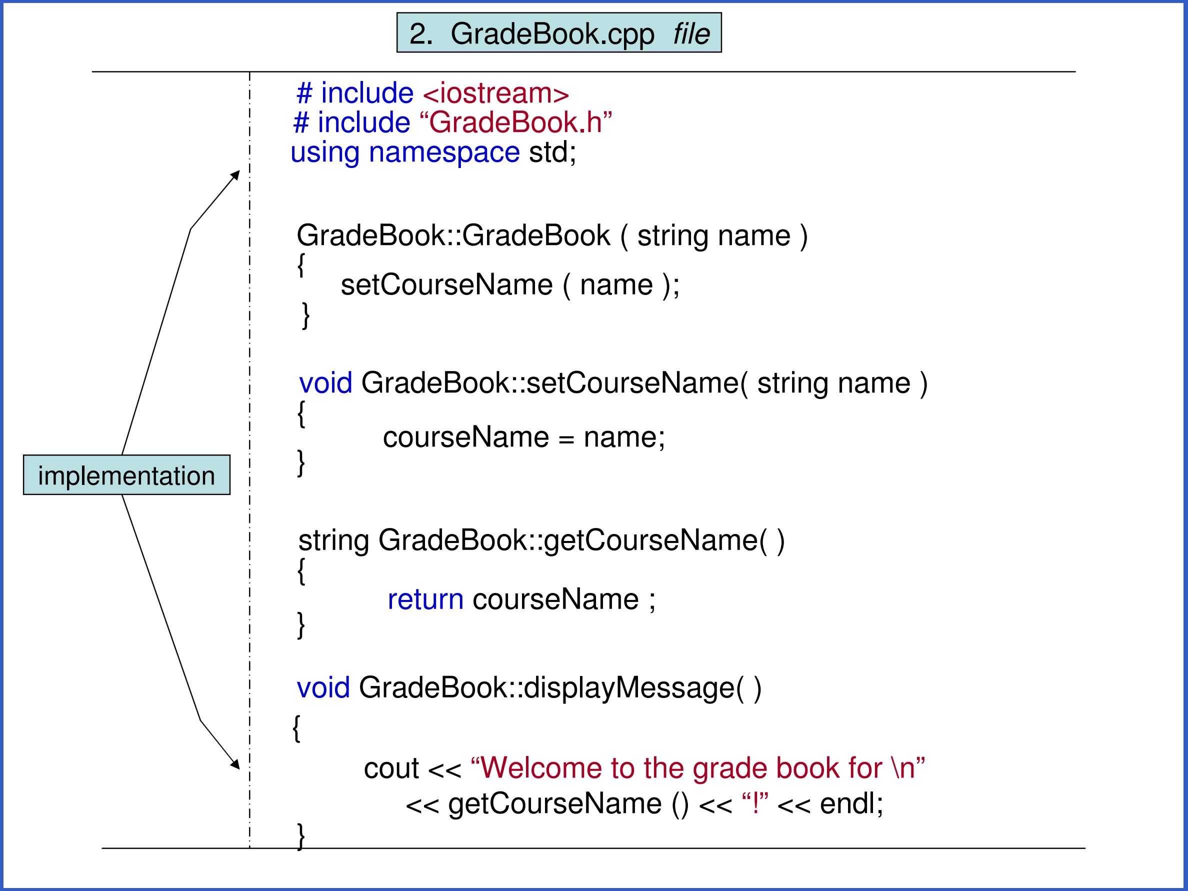
Separating *Interface* ***from*** *Implementation*

Example: class GradeBook IV



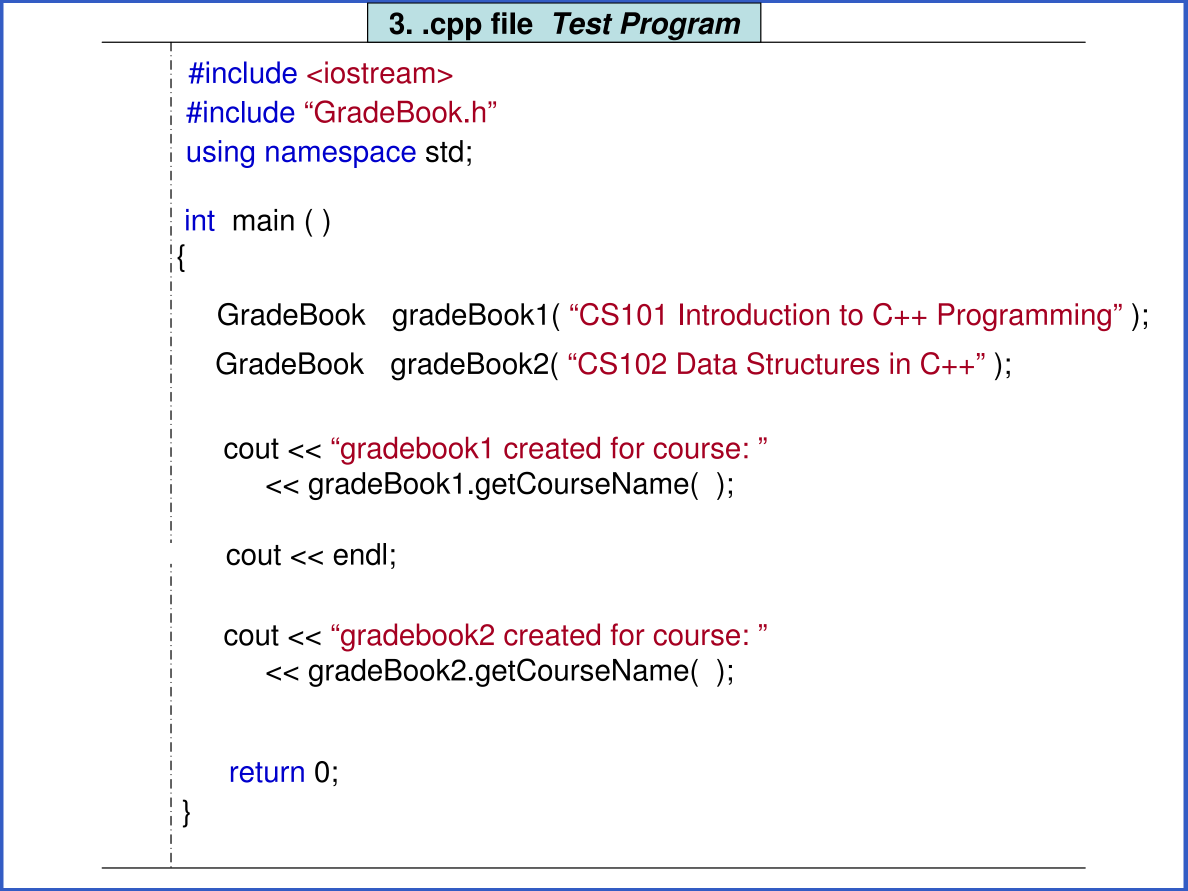
```
# include <string>

class GradeBook
 {
 public:
        GradeBook ( string name );      //constructor initializes course name
        void setCourseName( string name );  //function to set the course name
        string getCourseName( );        //function to get the course name
        void displayMessage( );         //display a welcome message
  private:
         string courseName;
 } ;
```

Example: class GradeBook IV



```
# include <iostream>
# include "GradeBook.h"
using namespace std;

GradeBook::GradeBook ( string name )
{
    setCourseName ( name );
}

void GradeBook::setCourseName( string name )
{
        courseName = name;
}

string GradeBook::getCourseName( )
{
        return courseName ;
}

void GradeBook::displayMessage( )
{
      cout << "Welcome to the grade book for \n"
          << getCourseName () << "!" << endl;
}
```

Example: class GradeBook IV



3. .cpp file ***Test Program***

```
#include <iostream>
#include "GradeBook.h"
using namespace std;

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

    cout << "gradebook1 created for course: "
        << gradeBook1.getCourseName(  );

    cout << endl;

    cout << "gradebook2 created for course: "
        << gradeBook2.getCourseName(  );


    return 0;
}
```

Exercise

Exercise

Exercise 1. Create a class called *Employee* that includes three pieces of information as data members — a first name (type *string*), a last name (type *string*) and monthly salary (type *int*). Your class should have a constructor that initializes the three data members. Provide a *set* function and *get* function for each data member. If the monthly salary is not positive, set it to 0. Write a test program that demonstrates class *Employee*'s capabilities. Create two *Employee* objects and display each object's *yearly* salary. Then give each *Employee* a 10 percent raise and display each *Employee*'s yearly salary again.

Exercise

Exercise 2. Create a class called *Date* that includes three pieces of information as data members — a month (type *int*), a day (type *int*) and a year (type *int*). Your class should have a constructor that initializes the three data members. For the purpose of this exercise, assume that the values provided for the year and day are correct, but ensure that the month value is in the range 1-12; if it is not, set the month to 1. Provide a *set* and a *get* function for each data member. Provide a member function *displayDate* that displays the month, day and year separated by forward slashes (/). Write a test program that demonstrates class *Date*'s capabilities.

Homework:

- Read Sec. 9.5 - 9.6
- Exercise 1, 2 in this slide
- Exercise 9.10