3.2 Defining a Class with a Member Function (cont.)

UML Class Diagram for Class `GradeBook`

- In the UML, each class is modeled in a UML class diagram as a *rectangle* with three *compartments*.
- Figure 3.2 presents a class diagram for class `GradeBook` (Fig. 3.1).
- The *top compartment* contains the class's name centered horizontally and in boldface type.
- The *middle compartment* contains the class's attributes, which correspond to data members in C++.
  - Currently empty, because class `GradeBook` does not yet have any attributes.
- The *bottom compartment* contains the class's operations, which correspond to member functions in C++.
- The UML models operations by listing the operation name followed by a set of parentheses.
- The plus sign (+) in front of the operation name indicates that `display-Message` is a public operation in the UML.

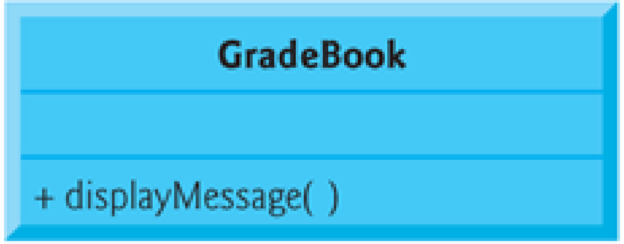

Fig. 3.2 | UML class diagram indicating that class GradeBook has a public displayMessage operation.

3.3 Defining a Member Function with a Parameter

- Car analogy
  - Pressing a car’s gas pedal sends a *message* to the car to perform a task—make the car go faster.
  - But how fast should the car accelerate? As you know, the farther down you press the pedal, the faster the car accelerates.
  - The message to the car includes *both* the *task to perform* and *additional information that helps the car perform the task*.
- Additional information that a function needs to perform its task is known as a parameter.

3.3 Defining a Member Function with a Parameter (cont.)

- Fig. 3.3 redefines class `GradeBook` (lines 9–18) with a `display-Message` member function (lines 13–17) that displays the course name as part of the welcome message.
  - The new version of `displayMessage` requires a *parameter* (`courseName` in line 13) that represents the course name to output.
- A variable of type `string` represents a string of characters.
- A string is actually an object of the C++ Standard Library class `string`.
  - Defined in header file `<string>` and part of namespace `std`.
  - For now, you can think of `string` variables like variables of other types such as `int`.
  - Additional `string` capabilities in Section 3.9.

```cpp
// Fig. 3.3: fig03_03.cpp
// Define class GradeBook with a member function that takes a parameter,
// create a GradeBook object and call its displayMessage function.
#include <iostream>
#include <string> // program uses C++ standard string class
using namespace std;

// GradeBook class definition
class GradeBook
{
public:
   // function that displays a welcome message to the GradeBook user
   void displayMessage( string courseName ) const
   {
      cout << "Welcome to the grade book for\n" << courseName << "!"
         << endl;
   } // end function displayMessage
}; // end class GradeBook

```

Fig. 3.3 | Define class GradeBook with a member function that takes a parameter, create a GradeBook object and call its displayMessage function. (Part 1 of 3.)

```
// function main begins program execution
int main()
{
   string nameOfCourse; // string of characters to store the course name
   GradeBook myGradeBook; // create a GradeBook object named 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

   // call myGradeBook's displayMessage function
   // and pass nameOfCourse as an argument
   myGradeBook.displayMessage( nameOfCourse );
} // end main
```

Fig. 3.3 | Define class GradeBook with a member function that takes a parameter, create a GradeBook object and call its displayMessage function. (Part 2 of 3.)

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

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

Fig. 3.3 | Define class `GradeBook` with a member function that takes a parameter, create a `GradeBook` object and call its `displayMessage` function. (Part 3 of 3.)

3.3 Defining a Member Function with a Parameter (cont.)

- Library function `getline` reads a line of text into a `string`.
- The function call `getline( cin, nameOfCourse )` reads characters (including the space characters that separate the words in the input) from the standard input stream object `cin` (i.e., the keyboard) until the *newline* character is encountered, places the characters in the `string` variable `nameOfCourse` and *discards* the newline character.
- When you press *Enter* while entering data, a newline is inserted in the input stream.
- The `<string>` header file must be included in the program to use function `getline`.

3.3 Defining a Member Function with a Parameter (cont.)

- Line 33 calls `myGradeBook`’s `displayMessage` member function.
  - The `nameOfCourse` variable in parentheses is the argument that is passed to member function `displayMessage` so that it can perform its task.
  - The value of variable `nameOfCourse` in `main` becomes the value of member function `displayMessage`’s parameter `courseName` in line 13.

3.3 Defining a Member Function with a Parameter (cont.)

- To specify that a function requires data to perform its task, you place additional information in the function’s parameter list, which is located in the parentheses following the function name.
- The parameter list may contain any number of parameters, including *none at all* to indicate that a function does *not* require any parameters.
- Each parameter must specify a type and an identifier.
- A function can specify multiple parameters by separating each parameter from the next with a comma.
- The number and order of arguments in a function call must match the number and order of parameters in the parameter list of the called member function’s header.
- The argument types in the function call must be consistent with the types of the corresponding parameters in the function header.

3.3 Defining a Member Function with a Parameter (cont.)

- The UML class diagram of Fig. 3.4 models class `GradeBook` of Fig. 3.3.
- The UML models a parameter by listing the parameter name, followed by a colon and the parameter type in the parentheses following the operation name.
- The UML has its own data types similar to those of C++.
- The UML is *language independent*—it's used with many different programming languages—

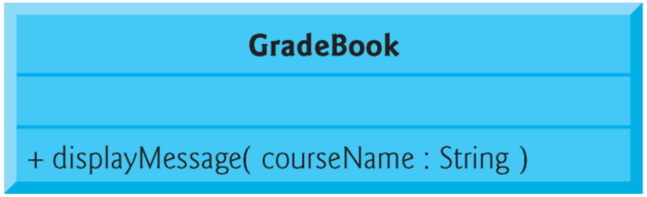

Fig. 3.4 | UML class diagram indicating that class GradeBook has a public displayMessage operation with a courseName parameter of UML type String.

3.4 Data Members, *set* Member Functions and *get* Member Functions

- Variables declared in a function definition's body are known as local variables and can be used only from the line of their declaration in the function to the closing right brace (}) of the block in which they're declared.
  - A local variable must be declared *before* it can be used in a function.
  - A local variable cannot be accessed *outside* the function in which it's declared.
  - *When a function terminates, the values of its local variables are lost.*

3.4 Data Members, *set* Member Functions and *get* Member Functions (Cont.)

- An object has attributes that are carried with it as it’s used in a program.
  - Such attributes exist throughout the life of the object.
  - A class normally consists of one or more member functions that manipulate the attributes that belong to a particular object of the class.
- Attributes are represented as variables in a class definition.
  - Such variables are called data members and are declared inside a class definition but outside the bodies of the class’s member-function definitions.
- Each object of a class maintains its own attributes in memory.

3.4 Data Members, *set* Member Functions and *get* Member Functions (Cont.)

- A typical instructor teaches several courses, each with its own course name.
- A variable that is declared in the class definition but outside the bodies of the class's member-function definitions is a *data member*.
- Every instance (i.e., object) of a class contains each of the class's data members.
- A benefit of making a variable a data member is that all the member functions of the class can manipulate any data members that appear