AAB University

Faculty of Computer Sciences

Object Oriented Programming

Week 4:

Introduction to Classes and Objects

Asst. Prof. Dr. **Mentor Hamiti**

mentor.hamiti@universitetiaab.com

Last Time?!

- Structure of a program
- Variables
- Memory Concepts
- Arithmetic
- Decision Making
- ……

Today

- Introduction to Classes and Objects
- Defining a Class with a Member Function
- Defining a Member Function with a Parameter
- Data Members, set Functions and get Functions
- Initializing Objects with Constructors
- Placing a Class in a Separate File for Reusability

Introduction

- The basic concepts of Object Oriented Programming are **Classes** and **Objects**

- Typically, programs written in C++ are composed by one function **main** and one or more classes

- Each Class contains:
  - **data members** and
  - **member functions**

- Introduction to Classes and Objects
- **Defining a Class with a Member Function**
- Defining a Member Function with a Parameter
- Data Members, set Functions and get Functions
- Initializing Objects with Constructors
- Placing a Class in a Separate File for Reusability

Example 1

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

```
//Example 1
#include <iostream>
using namespace std;


// 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 class GradeBook


int main()
{
    GradeBook myGradeBook;           // create a GradeBook object named myGradeBook
    myGradeBook.displayMessage();    // call object's displayMessage function

    return 0;
}
```

Defining a Class with a Member Function

- The class definition begins with the keyword **class** followed by the class name **GradeBook**
  - *By convention, the name of a user-defined class begins with a capital letter, and for readability, each subsequent word in the class name begins with a capital letter*
- Every class's body is enclosed in a pair of left and right braces **{** and **}**
- The class definition terminates with a semicolon **;**

Common Programming Error 3.1

Forgetting the semicolon at the end of a class definition is a syntax error.

Defining a Class with a Member Function

- Function main is always called automatically when you execute a program
- Most functions do not get called automatically
- You must call member function **displayMessage** explicitly to tell it to perform its task
- The access-specifier label public: contains the keyword **public** is an access specifier
  - Indicates that the function is "**available to the public**" that is, it can be called by other functions in the program (such as main), and by member functions of other classes
  - Access specifiers are always followed by a colon (**:**).

Defining a Class with a Member Function

- Each function in a program performs a task and may return a value when it completes its task

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

- Keyword **void** to the left of the function name **displayMessage** is the function's return type
  - Indicates that **displayMessage** will not return any data to its calling function when it completes its task
- By convention, function names use a lowercase first letter

Defining a Class with a Member Function

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

- The parentheses after the member function name indicate that it is a **function**
  - Empty parentheses indicate that a member function does not require additional data to perform its task
- The first line of a function definition is commonly called the function header
- Every function's body is delimited by braces { and }
- The function body contains statements that perform the function's task

Testing Class GradeBook

- Typically, you cannot call a member function of a class until you create an **object** of that **class**

- First, create an object of class GradeBook called myGradeBook
  - The variable's type is GradeBook
  - The compiler does not automatically know what type GradeBook is—it's a user-defined type
  - Tell the compiler what GradeBook is by including the class definition
  - Each class you create becomes a new type that can be used to create objects

Testing Class GradeBook

- Create an **Object** of class GradeBook called **myGradeBook**

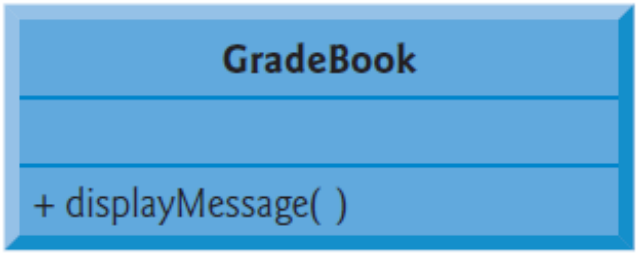
```
GradeBook myGradeBook;
myGradeBook.displayMessage();
```

- Call the member function displayMessage- by using variable **myGradeBook** followed by the dot operator **.** the function name **displayMessage** and an empty set of parentheses **( )**
- Causes the displayMessage function to perform its task

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

UML Class Diagram for Class GradeBook

- In the UML, each class is modeled in a **UML class diagram** as a rectangle with three compartments:



  - *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 plus sign + in front of the operation name indicates that displayMessage is a public operation in the UML

- Introduction to Classes and Objects
- Defining a Class with a Member Function
- Defining a Member Function with a Parameter
- Data Members, set Functions and get Functions
- Initializing Objects with Constructors
- Placing a Class in a Separate File for Reusability

Example 2

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

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

```
//Example 2
#include <iostream>
#include <string>       // program uses C++
using namespace std;

// GradeBook class definition
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;
    }
};

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

    cout << "Please enter the course name:" << endl;
    getline(cin, nameOfCourse ); // read a course name with blanks
    myGradeBook.displayMessage(nameOfCourse);

    return 0;
}
```

Defining a Member Function with a Parameter

```
#include <string>
```

- 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**

Defining a Member Function with a Parameter

```
getline(cin, nameOfCourse );
```

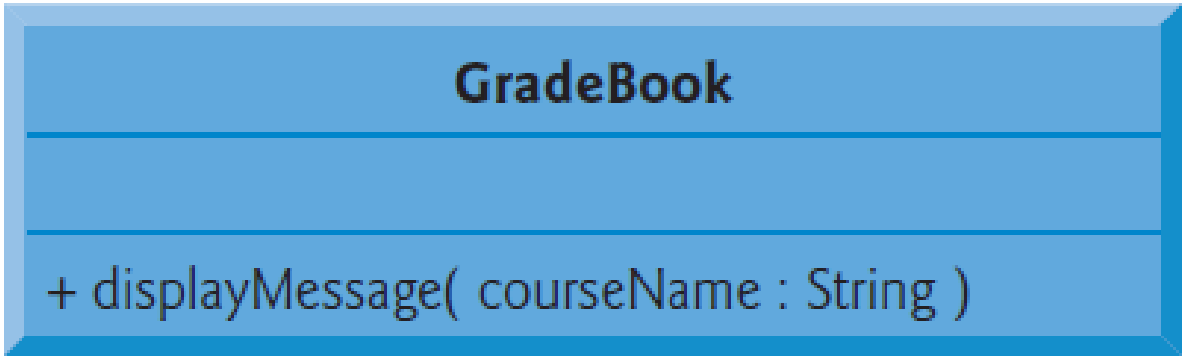
- Library function getline reads a line of text into a string
  - The function call **getline(cin, nameOfCourse)** reads characters (including the space characters that separate the words in the input) from the standard input stream object **cin** (the keyboard) until the newline character is encountered
  - When Enter is pressed 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**

Defining a Member Function with a Parameter

- Functions with parameters requires data to perform its task
  - 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

UML Class Diagram for Class GradeBook (Ex.2)

- The UML has its own data types similar to those of C++
  - *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 is language independent
  - *It's used with many different programming languages, so its terminology does not exactly match that of C++*

- Introduction to Classes and Objects
- Defining a Class with a Member Function
- Defining a Member Function with a Parameter
- Data Members, set Functions and get Functions
- Initializing Objects with Constructors
- Placing a Class in a Separate File for Reusability

Data Members, set Functions and get 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*

Data Members, set Functions and get Functions

- 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

Data Members, set Functions and get Functions

- 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 in the class definition

Example 3

```
//Example 3
#include <iostream>
#include <string> // program uses C++ standard string class
using namespace std;

// GradeBook class definition
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

    // 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 GradeBook
}; // end class GradeBook
```

Example 3 (cont.)

```
// 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

    // 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:" << endl;
    getline( cin, nameOfCourse ); // read a course name with blanks

    myGradeBook.setCourseName( nameOfCourse ); // set the course name
    cout << endl; // outputs a blank line

    myGradeBook.displayMessage(); // display message with new course name
} // end main
```

```
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!
```

Data Members, set Functions and get Functions

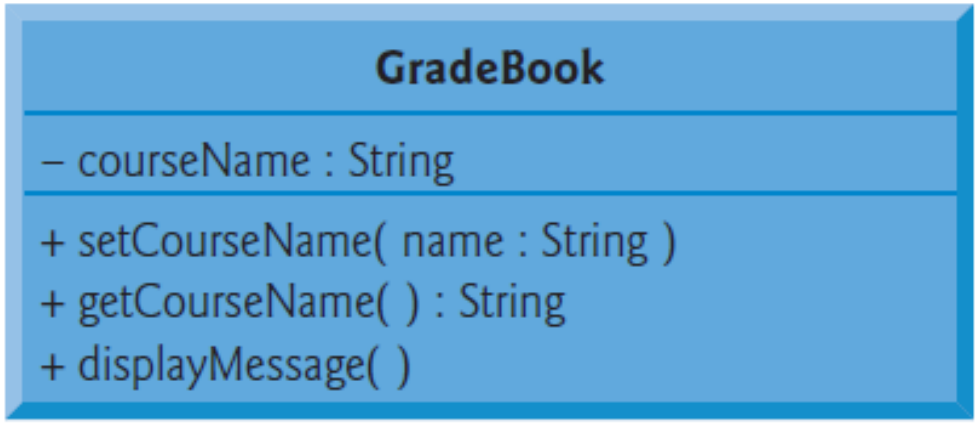
- Most data-member declarations appear after the access-specifier label **private:**
- Like **public**, keyword **private** is an access specifier
- Variables or functions declared after access specifier private (*and before the next access specifier*) are accessible only to member functions of the class for which they're declared
- The default access for class members is private so all members after the class header and before the first access specifier are private
- The access specifiers public and private may be repeated, but this is unnecessary and can be confusing

Data Members, set Functions and get Functions

- Declaring data members with access specifier **private** is known as **data hiding**
- When a program creates an **object**, its data members are **encapsulated** (hidden) in the object and can be accessed only by member functions of the object's class
- Classes often provide public member functions to allow clients of the class to **set** (*i.e., assign values to*) or **get** (*i.e., obtain the values of*) private data members
  - *These member function names need not begin with **set** or **get**, but this naming convention is common*
  - *Set functions are also sometimes called **mutators** (because they mutate, or change, values), and get functions are also sometimes called **accessors** (because they access values)*

UML Class Diagram for Class GradeBook (Ex.3)

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



  - *The UML represents data members as attributes by listing the attribute name, followed by a colon and the attribute type*

- Introduction to Classes and Objects
- Defining a Class with a Member Function
- Defining a Member Function with a Parameter
- Data Members, set Functions and get Functions
- Initializing Objects with Constructors
- Placing a Class in a Separate File for Reusability

Initializing Objects with Constructors

- Each class can provide one or more **constructors** that can be used to initialize an object of the class when the object is created
- A constructor is a special member function that must be defined with the **same name** as the **class**, so that the compiler can distinguish it from the class's other member functions
- An important difference between constructors and other functions is that constructors *cannot* ***return*** *values*, so they cannot specify a return type (*not even* ***void***)
- Normally, constructors are declared **public**

Initializing Objects with Constructors

- C++ automatically calls a constructor for each object that is created, which helps ensure that objects are initialized properly before they're used in a program
- The constructor call occurs when the object is created
- If a class does not explicitly include constructors, the compiler provides a **default constructor** with no parameters

Example 4

```
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

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

    // function to get the course name
    string getCourseName()
    {
    return courseName; // return object's courseName
    } // end function getCourseName

    // display a welcome message to the GradeBook user
    void displayMessage()
    {
    // call getCourseName to get the courseName
        cout << "Welcome to the grade book for\n" <<getCourseName()<< "!" << endl;
    } // end function displayMessage

private:
    string courseName; // course name for this GradeBook
}; // end class GradeBook
```

Example 4 (cont.)

```
//Example_4
# include <iostream>
# include <string>
using namespace std;

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

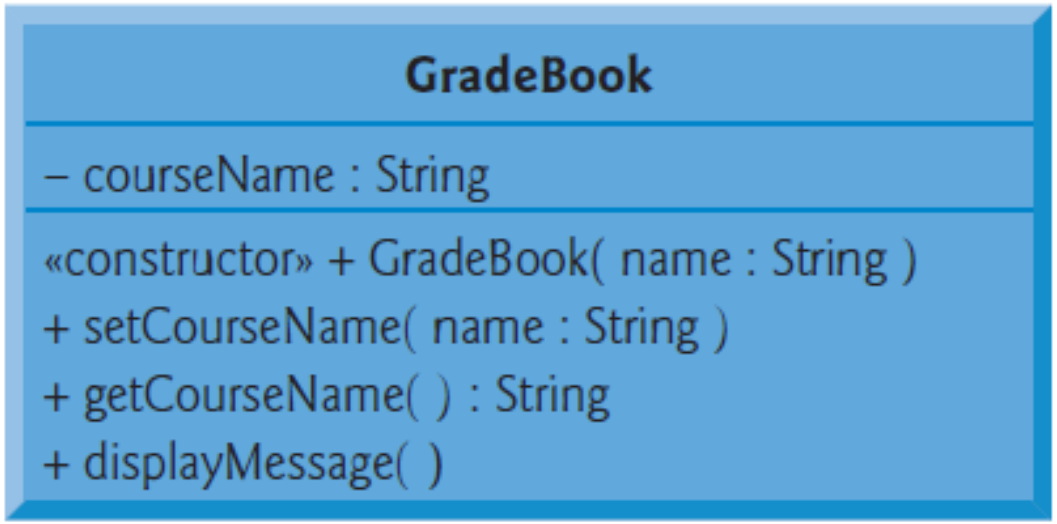
```
gradeBook1 created for course: CS101 Introduction to C++ Programming
gradeBook2 created for course: CS102 Data Structures in C++
```

Initializing Objects with Constructors

- A constructor specifies in its parameter list the data it requires to perform its task
- When you create a new object, you place this data in the parentheses that follow the object name
- Any constructor that takes no arguments is called a default constructor
- A class gets a default constructor in one of several ways:
  - *The compiler implicitly creates a default constructor in every class that does not have any user-defined constructors*
  - *You explicitly define a constructor that takes no arguments*
  - *If you define any constructors with arguments, C++ will not implicitly create a default constructor for that class*

UML Class Diagram for Class GradeBook (Ex.4)

- Like operations, the UML models constructors in the third compartment of a class in a class diagram
- To distinguish a constructor from a class's operations, the UML places the word "constructor" between guillemets (« and ») before the constructor's nam



- Introduction to Classes and Objects
- Defining a Class with a Member Function
- Defining a Member Function with a Parameter
- Data Members, set Functions and get Functions
- Initializing Objects with Constructors
- Placing a Class in a Separate File for Reusability

Placing a Class in a Separate File for Reusability

- One of the benefits of creating class definitions is that, when packaged properly, our classes can be reused by programmers—potentially worldwide
- Programmers who wish to use our GradeBook class cannot simply include the file from another program
  - *Function main begins the execution of every program, and every program must have exactly one main function*
- When building an object-oriented C++ program, it's customary to define reusable source code (*such as a class*) in a file that by convention has a **.h filename** extension—known as a **header**
- Programs use #include preprocessing directives to include header files and take advantage of **reusable software components**

Example 5 (cont.)

```
//GradeBook.h File
# include <iostream>
# include <string>
using namespace std;
// 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

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

    // function to get the course name
    string getCourseName()
    {
    return courseName; // return object's courseName
    } // end function getCourseName

    // display a welcome message to the GradeBook user
    void displayMessage()
    {
    // call getCourseName to get the courseName
        cout << "Welcome to the grade book for\n" <<getCourseName()<< "!" << endl;
    } // end function displayMessage
private:
    string courseName; // course name for this GradeBook
}; // end class GradeBook
```

Example 5

```
//Example_5
# include <iostream>
# include "GradeBook.h"
using namespace std;

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

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

Placing a Class in a Separate File for Reusability

- A **# include** directive instructs the C++ preprocessor to replace the directive with a copy of the contents of **GradeBook.h** before the program is compiled

- When the source-code file is compiled, it now contains the GradeBook class definition (*because of the #include*), and the compiler is able to determine how to create GradeBook objects and see that their member functions are called correctly

- Now that the class definition is in a header file (*without a **main** function*), we can include that header in any program that needs to reuse our GradeBook class

Placing a Class in a Separate File for Reusability

- Notice that the name of the **GradeBook.h** header file in is enclosed in quotes " " rather than angle brackets < >
  - *Normally, a program's source-code files and user-defined header files are placed in the same directory*
  - *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 (e.g., <iostream>), it assumes that the header is part of the C++ Standard Library and does not look in the directory of the program that is being preprocessed*

- Questions?!

mentor.hamiti@universitetiaab.com