CEE432/CEE532/MAE541
Developing Software for Engineering Applications

Lecture 6: First Look at Objects and Classes (Objects 101)

(Chapter 7)

A New of Thinking

- User-defined data types
- The thought process to create these user-defined types is known as **data abstraction**. Data abstraction is defined as separating the overall properties of a data type from its implementation. The mechanism to implement the data abstraction is called **encapsulation**.

What are classes?

- Entities that have
  - *attributes* (defined as having properties)
  - *behavior* (capabilities to do something)
- Classes help us define our own data types and manipulate them

What is an object?

- An instance of a class is an object.
- All objects must be unique and uniquely identified.
- In C++ language, an object is a variable associated with a class.

Components of a Class

- Syntax of a class definition

```
class class_name
{
  public:
  …
  private:
  …
  protected:
  …
};
```

Components of a Class

- Constructor (**ctor**)
  - The name of this function is the class name itself.
  - This function is called once and only once when the (class object) is defined.
  - There is no return type (incl. `void`).
  - We can have overloaded **ctor**s
  - This is the place to initialize all class variables.
  - **Warning**: Definition of this function is optional.

Components of a Class

- Destructor (**dtor**)
  - The name of this function is **`~classname`**.
  - This function is called once and only once when the class-related object goes out of scope.
  - There is no return type (incl. `void`).
  - We cannot have overloaded **dtor**s
  - This is the place to “clean up” resources.
  - **Warning**: Definition of this function is optional.

Components of a Class

- Member Variables
  - Declared in the class definition.
  - These usually are the C++ built in data type variables
  - Warning: These variables have class wide visibility.
  - Naming convention **`m_variablename`**

Components of a Class

- Member Functions
  - Declared in the class definition.
  - These follow the usual function properties (re. Chapter 4)
  - Are available only via class-associated objects
  - Have access to all member variables

Why Encapsulation?

- **Encapsulation** also referred to information hiding, refers to the technique by which data attributes and behavior-related operations are linked together such that the data can be manipulated only through these operations not directly. The program or code that stores the data and implements the behavior is called the **server code**.

Class Properties

- Public variables and functions
  - These variables and functions are available outside the class
  - A trait of encapsulation

Class Properties

- Private Variables and Functions
  - These variables and functions are available ONLY within the class
  - A trait of encapsulation

Class Definition

- Server code
  - Header File – mypoint.h
  - Source File (server code) – mypoint.cpp
- All function definitions are preceded by the scope resolution operator `::`
- Syntax

```
returntype classname::functionname (…)
```

Accessor Functions

- These are public functions to **access** the values of some or all of the member variables. The access is NOT directly done by the client code. The server code assists this access.
- We will call these function(s) **`GetValues`**

Modifier Functions

- These are public functions to **modify** the values of some or all of the member variables. This modification is NOT done directly by the client code. The server code assists this modification.
- We will call these function(s) **`SetValues`**

Let’s Write the Server Code

- Example using **`CMyPoint`** class
- Keep thinking about how you would write the code if you did not know OOP.

Objects: Using Classes

- Client code

```
#include "mypoint.h"
….
CMyPoint P1;
CMyPoint P2(13.3f, -4.5f);
CMyPoint P3(108.45f, fYCoor);
```

- All function access is via the member selection operator `.`

Let’s Write the Client Code

- Example using **`CMyPoint`** class
- Keep thinking about how you would write the code if you did not know OOP.

Enhancing Capabilities

- What functionalities would you like to add to the class?
- Will adding these functionalities by easy?
- Will it affect existing code (client code)?

Copy Constructor

- Used to make a copy of an object using another object

```
CPoint P1 = P2;
CPoint P1 (P2);
```

- Note C++ automatically supplies the **`=`** operator for use with any class

When are Copy Ctors Used?

- When a declaration is made with *initialization from another object*.
- Parameters are passed by value.
- An object is returned by a function.

Making the Enhancements

- Adding `const` to arguments and functions
- Copy `ctor`
- New functionalities via new member functions
  - Distance of a point from another point
  - Translating a point
- Extending the point class to support points in space

Class Composition

•Objects can be part of another object

```
class CSensor
{
…
CTime     m_TimeStamp;
CPoint    m_Location;
…
};
```

More on std::string class

- Declaration and initialization

```
string szInput1, szInput2;
string szHeader = “How’s that”;
string szPrompt (“Input: “)
string szCpyPrt (szPrompt);
```

- Declaration and initialization
  - Need “ ..” even if initializing string with one character

More on std::string class

- Assigning values to strings

```
szName = szOldName;
szFullName = szFirstName +
  szLastName;
szFullName = szLastName + “, ” +
  szFirstName;
szFullName += szOldName;
```

- Following is invalid

```
szName = “John “ + “Doe”;
```

More on std::string class

- Accessing individual components
  - Done via the **[ ]** operator

```
szName = “Arizona”;
szName[3] = ‘6’;
```

More on std::string class

- Using as a function parameter

```
void Display (const
  std::string& szHeader);
void GetName (std::string&
  szName);
```

More on std::string class

- Comparing strings
  - Achieved using the logical operators such as **`==, !=, <, >`** etc.

More on std::string class

- Working with substrings

```
string szFirstPart, szLastPart;
string szTitle = “Vector A”;
szFirstPart = szTitle.substr
  (0, 6); // extracts Vector
szLastPart = szTitle.substr (7,
  1); // extracts A
```

More on std::string class

- "Finding" member functions
  - find
  - find_first_of
  - find_last_of
  - rfind
  - find_first_not_of
  - find_last_not_of

More on std::string class

- Inserting strings

```
string szDigits = "123321";
string szAlphabets = "abcd";
szDigits.insert (3, szAlphabets);
```

More on std::string class

- Handling upper and lower cases

```
#include <cctype>
toupper (variable);
tolower (variable);
```

Summary

- Class definitions form the foundation of OOP
- Quick Facts
  - Constructor
  - Destructor
  - Member variables
  - Member functions
  - **`public`**
  - **`private`**

Summary

- Quick Facts
  - Function definitions require scope resolution operator :: (in the server code)
  - Function access requires member selection operator .
  - Copy Constructor
  - Composition

Summary

- *Programming Style Tip 7.1: Define a default constructor and a destructor*
- *Programming Style Tip 7.2: Define the copy constructor*
- *Programming Style Tip 7.3: Practice defensive programming*