References

Chapter 13 – Structured Data

Starting Out With C++. From Control Structures through Objects (Eighth Edition)

Interview Question:

- What are the differences between a class and its object?

What is a class?

A class is used to specify the form of an object and it combines (1) **data members** and (2) **member functions** for manipulating that data into one neat package. The data and functions within a class are called members of the class. The relationship between class and object is somewhat like this:

Class **:** Object = Blueprint **:** House

This indicates that we can **instantiate** an object out of a class in the similar way like that we can build a house out of a blueprint. Using the same blueprint we can build unlimited amount of houses. In the similar way we can instantiate unlimited amount of objects.

The parts of a class

Data members of an object are the **properties** (or **attributes**) of that object, and the member functions are the **operations** which can be performed by the object. As shown in the following table, a **class** consists of six parts.

Part 1:	They are the properties of the class. The three types of

Data members	data member access specifiers are: **public**, **private** and **protected**. (These specifiers will be explained in the sections below.)
Part 2: **Constructors**	The constructors do not actually construct the object. Instead, they are the functions which allow us to **initialize data members**.
Part 3: **Setters**	They are **public** member functions which set the values of the private data members.
Part 4: **Getters**	They are **public** member functions which return the values of the private data members.
Part 5: **Destructor**	The destructor does not actually destruct the object. Instead, it is the function which allows us **do the cleanup work, such as releasing dynamically allocated memories**. There can be only one destructor for each class, which is the one provided by the author or a default destructor automatically provided by the system.
Part 6: **Other member functions**	The member functions which are not in any the four categories, i.e. constructors, setters, getters, and destructor, listed above.

Exercise 1:

The following program gives us a brief overview of a class. When using this sample program, you should first separate them into three files:

main.cpp
Rectangle.h
Rectangle.cpp

Include all the three files in a project.

```
////// Beginning of Rectangle.h
class Rectangle
{
private:
```

```
   int serialNumber;
   double width;
   double height;

public:
   Rectangle(int, int);
   ~Rectangle();
   void setWidth(double);
   void setHeight(double);
   void setSerialNumber(int);
   double getWidth() const;
   double getHeight() const;
   int getSerialNumber() const;
};
////// End of Rectangle.h

////// Beginning of Rectangle.cpp
#include "Rectangle.h"


   Rectangle::Rectangle(int w, int h)
   {
      width= w;
      height = h;
   }
   Rectangle::~Rectangle()
   {

   }
   void Rectangle::setWidth(double w)
   {
      width = w;
   }

   void Rectangle::setHeight(double h)
   {
      height = h;
```

```
   }
   double Rectangle::getWidth() const
   {
      return width;
   }

    double Rectangle::getHeight() const
   {
      return height;
   }

   void Rectangle::setSerialNumber(int s)
   {
      serialNumber = s;
   }
   int Rectangle::getSerialNumber() const
   {
      return serialNumber;
   }
////// End of Rectangle.cpp



////// Beginning of main.cpp
#include <iostream>

using namespace std;
#include "Rectangle.h"

const int SIZE = 21;

int main()
{
   Rectangle rect1(2, 4);

   rect1.serialNumber(1001);
   cout << rect1.getSerialNumber();
```

```
}
////// End of main.cpp
```

Question 1: Which parts of class Rectangle are data members? Are they public, private or protected?

Question 2: Which functions(s) of class Rectangle is/are constructors, setters, getters or destructor?

Question 3: Delete the constructors and modify the code in main() to make the program work. Can a class have no constructor?

Question 3: Add the following two lines to a constructor which takes no parameter. Create 21 object without providing any width or height. Display the width and height of these objects to see they are.

```
width= 5;
height = 7;
```

Discussions:

- Does a constructor itself construct any object? Why?
- How many constructors can a class have?
- Can a class have no constructor?
- How many parameter(s) does the default constructor have?
- How many destructor can be in a class?
- How many parameter(s) does a destructor have?

Constructors

Constructors are the member functions of a class. One of the constructors is **automatically** called when an object is constructed. There can be multiple

constructors in a class. The constructor, which is called, is the one which has the same number of parameters and the type of each parameter match that of its corresponding argument in the caller function. For example, the following statement:

```
Rectangle rect(4, 5);
```

will invoke a construct which has two integer parameter, and

```
Rectangle rect(4.0, 5.0);
```

will invoke a constructor with two double parameter, while .

```
Rectangle rect;
```

will invoke the one with no parameter.

With the provided values, such as 4 and 5, the constructor can initialize the object properties in the following way.

```
Rectangle::Rectangle(int w, int h)
{
   width= w;
   height = h;
}
```

Default values can be provided. If the default values for width and height are 1, then the values can be added to the parameters like this.

```
Rectangle::Rectangle(int w = 1, int h = 1)
{
   width= w;
```

```
    height = h;
}
```

In case that a Rectangle object is constructed without any initial value, this constructor is called and the default width and height are provided. More precisely, the constructor above actually serves for three different types of constructions: the one with two arguments, one argument and no argument.

One thing which we need to pay special attention is that the default value setting should be started from the end to the beginning. What this means is that the following constructor, which provides only one default value for the second parameter, is acceptable.

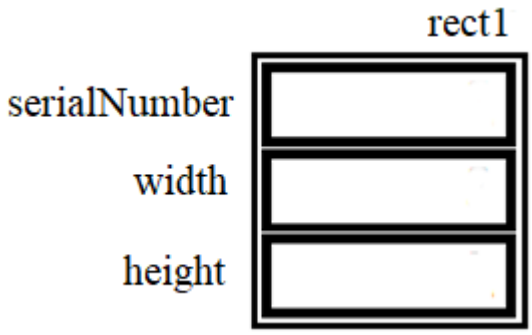
```
Rectangle::Rectangle(int w, int h = 1)
{
    width= w;
    height = h;
}
```

But the following constructor, which provides only one default value for the first parameter, is NOT acceptable.

```
    Rectangle::Rectangle(int w = 1, int h)
{
    width= w;
    height = h;
}
```

The reason for this is obvious. If both constructors exist in the system, then the system cannot tell which constructer to use when the caller function provides only one argument.

A constructor is called after the object is constructed. At the moment the object already exists in the memory. Let us use the Rectangle as example, the memory allocation looks like this:



Since serialNumber is an integer and the other two are double, it should occupy 28 bytes of memory. We can use sizeof() function to check the size of the object.

Discussions:

- Why the member functions in the class do not take any memory space?

Destructor

A class allows only one destructor and it does not have any parameter. The destructor is called before the object is destructed. Since the object will be destructed right away and all the values of non-static data members will be destructed, there is no need to modify these members. The purpose of the destructor is for de-allocating the dynamically allocated memory. Code example will be provided in the Manager Class chapter.

Access specifiers: public, private and protected

There are 3 access specifiers for a class/struct/Union in C++. These access specifiers define how the members of the class can be accessed. Of course, any

member of a class is accessible within that class (Inside any member function of that same class). Moving ahead to type of access specifiers, they are:

public	The members declared as **public** are accessible from outside the Class through an object of the class.
private	These members are only accessible from within the class. No outside Access is allowed.
protected	The members declared as protected are **NOT** accessible from outside the class **BUT** only in a class derived from it.

Exercise 2:

The following program intends to set the height and width of the Rectangle object Rect1. Can it compile? If not, could you use at least two different ways to modify the program so that it can compile and the properties can be set?

```
#include <iostream

using namespace std;

class Rectangle
{
private:
   int serialNumber;
   double width;
   double height;

public:
   setWidth(double w)
   {
      width = w;
   }

   void setHeight(double h)
   {
```

```
        height = h;
    }
    double getWidth() const
    {
        return width;
    }

     double getHeight() const
    {
        return height;
    }

    void setSerialNumber(int s)
    {
        serialNumber = s;
    }
    int getSerialNumber() const
    {
        return serialNumber;
    }
};

int main()
{
    Rectangle Rect1;

    Rect1.height = 10;
    Rect1.width = 5;
}
```

Encapsulation - Setters and getters

What's the advantage of using getters and setters - that only get and set - instead of simply using public fields for those variables?

Setters and getters provide one extra layer of protection to the sensitive data. Here are the some of the reasons I am aware of:

- Getters and setters provide a centralized place for controlling the behaviors of the field (data member).
- Encapsulation of behavior associated with getting or setting the property - this allows additional functionality (like validation) to be added more easily later.
- Providing a debugging interception point for when a property changes at runtime - debugging when and where a property changed to a particular value can be quite difficult without this in some languages.

Note: Key word expression is a very efficient way on learning programming concepts. When using this method, we pick up one or a few key words in the statements and place it in the front of the statement. It helps us to easily organize our thoughts and memorize the points. Could you rewrite the three points above by using this method?

- **Centralized control** - Getters and setters provide a centralized place for controlling the behaviors of the field (data member).
- **Validation** - Encapsulation of behavior associated with getting or setting the property - this allows additional functionality (like validation) to be added more easily later.
- **Assert** - Providing a debugging interception point for when a property changes at runtime - debugging when and where a property changed to a particular value can be quite difficult without this in some languages.

Class versus struct

Class and struct are just like twin brothers. They are named Class Smith and Struct Smith. The only difference between the two is that by default all members of a class are private, while members of a struct are public.