Dr. Neamah E. Kadhim

2.1 Object-Oriented Programming Lecture Five

Object-oriented design (OOD) is a widely used programming methodology. In OOD, the first step in the problem-solving process is to identify the components called **objects**, which form the basis of the solution, and to determine how these objects interact with one another. After identifying the objects, the next step is to specify for each object the **relevant data** and **possible operations** to be performed on that data. An object combines data and operations on the data into a single unit.

In OOD, the final program is a collection of interacting objects. A programming language that implements OOD is called an **object-oriented programming (OOP)** language.

Because an object consists of **data** and **operations** on that data, before you can design and use objects, you need to learn how to represent data in computer memory, how to manipulate data, and how to implement operations.

To create operations, you write algorithms and implement them in a programming language. Because a data element in a complex program usually has many operations, **to separate operations from each other** and to use them effectively and in a convenient manner, you use **functions** to implement algorithms.

Finally, to work with objects, you need to know how to combine data and operations on the data into a single unit. In C++, the mechanism that allows you to combine data and operations on the data into a single unit is called a class.

A **class** embodies a meaningful grouping of characteristics and behaviors. The term **object** (instance) may also (and more often) be used to describe a specific item in such a grouping. Let's consider some examples:

Class	Object (Instance)
student	Specific student in computer science department
university	University of Baghdad
bank	National bank

2.2 Understanding object-oriented concepts

The key object-oriented concepts are *encapsulation* and *information hiding*. Incorporating these interrelated ideas into your design will provide the basis for writing more easily modifiable and maintainable programs.

The grouping of meaningful characteristics (attributes) and behaviors (operations) that operate on those attributes, bundled together in a single unit, is known as **encapsulation.**

Information hiding refers to the process of *abstracting* the details of performing an operation into a class method. That is, the user needs only to understand which operation to utilize and its overall purpose; the implementation details are hidden within the method (the function's body). In this fashion, changing the underlying implementation (method) will not change the operation's interface. Information hiding can additionally refer to keeping the underlying implementation of a class' attributes hidden. Information hiding is a means to achieve proper encapsulation of a class. **A properly encapsulated class will enable proper class abstraction and thus the support of OO designs**.

2.3 Understanding class in C++

When you define a C++ Class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object. In C++, class is a reserved word, and it defines only a data type; no memory is allocated. It announces the declaration of a class.

Class class_name { **Class body};**	**Class class_name {** **Class body} object name(s);**

The members of a **class** are classified into three categories: **private, public,** and **protected.** This lecture mainly discusses the first two types**, private** and **public**. In C++, **private**, **protected**, and **public** are reserved words and are called member access specifiers.

Following are some facts about public and private members of a class:

1) A public member is accessible outside of the class.

EX: Class with only public data

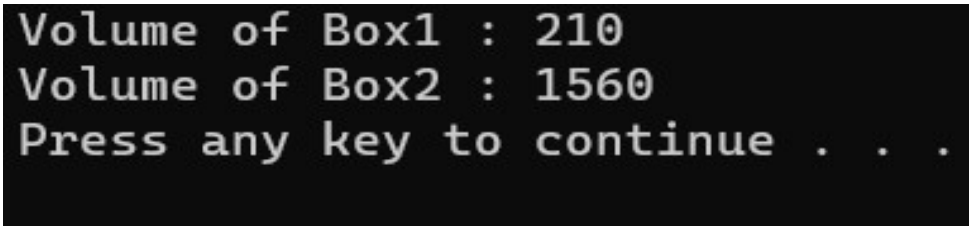
```
#include <iostream>
using namespace std;
class Box {
    public:
    double length; // Length of a box
    double breadth; // width of a box
```

```
        double height; // Height of a box
        };
int main() {
        Box Box1; // Declare Box1 of type Box
        Box Box2; // Declare Box2 of type Box
        double volume = 0.0; // Store the volume of a box here
        // box 1 specification
        Box1.height = 5.0;
        Box1.length = 6.0;
        Box1.breadth = 7.0;
        // box 2 specification
        Box2.height = 10.0;
        Box2.length = 12.0;
        Box2.breadth = 13.0;
        // volume of box 1
        volume = Box1.height * Box1.length * Box1.breadth;
        cout << "Volume of Box1 : " << volume <<endl;
        // volume of box 2
        volume = Box2.height * Box2.length * Box2.breadth;
        cout << "Volume of Box2 : " << volume <<endl;
        return 0;
}
```

EX: Class with public data and method to set the data.

```
#include <iostream>
using namespace std;
class Box {
public:
double length; // Length of a box
double breadth; // Breadth of a box
double height; // Height of a box
void setDim(double l,double b, double h)
    {    length=l;
         breadth=b;
         height=h;    }
};
int main() {
                Box Box1; // Declare Box1 of type Box
                Box Box2; // Declare Box2 of type Box
                double volume = 0.0; // Store the volume of a box here
                // box 1 specification
                Box1.setDim(5.0,6.0,7.0);
                // box 2 specification
                Box2.setDim(10.0,12.0,13.0);
                // volume of box 1
                volume = Box1.height * Box1.length * Box1.breadth;
```

```
cout << "Volume of Box1 : " << volume <<endl;
// volume of box 2
volume = Box2.height * Box2.length * Box2.breadth;
cout << "Volume of Box2 : " << volume <<endl;
return 0;}
```



EX: Class with public data and methods.

```
#include <iostream>
using namespace std;
class Box {
public:
double length; // Length of a box
double breadth; // Breadth of a box
double height; // Height of a box
void setDim(double l,double b,double h)
   {   length=l;
       breadth=b;
       height=h; }
double volume()
{
        double v;
        v=height * length * breadth;
```

```
        return v;}
};
int main() {
        Box Box1; // Declare Box1 of type Box
        Box Box2; // Declare Box2 of type Box
        double v = 0.0; // Store the volume of a box here
        // box 1 specification
        Box1.setDim(5.0,6.0,7.0);
        // box 2 specification
        Box2.setDim(10.0,12.0,13.0);
        // volume of box 1
        v = Box1.volume();
        cout << "Volume of Box1 : " << v <<endl;
        // volume of box 2
        v = Box2.volume();
        cout << "Volume of Box2 : " << v <<endl;
        return 0;
}
```

EX: Class with public data and methods(the implementation of methods outside the declaration of class)

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

```
class Box {
public:
double length; // Length of a box
double breadth; // Breadth of a box
double height; // Height of a box
void setDim(double ,double ,double );
double volume();
};
void Box::setDim(double l,double b,double h)
    {    length=l;
         breadth=b;
         height=h; }
double Box::volume()
{  double v;
  v=height * length * breadth;
  return v;}
int main() {
Box Box1; // Declare Box1 of type Box
Box Box2; // Declare Box2 of type Box
double v = 0.0; // Store the volume of a box here
// box 1 specification
Box1.setDim(5.0,6.0,7.0);
// box 2 specification
Box2.setDim(10.0,12.0,13.0);
```

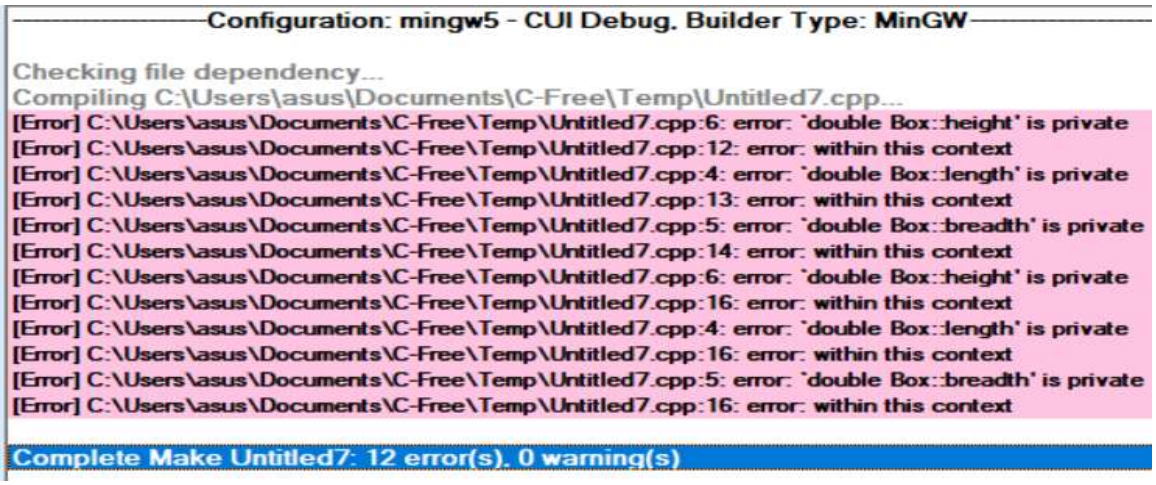
```
// volume of box 1
v = Box1.volume();
cout << "Volume of Box1 : " << v <<endl;
// volume of box 2
v = Box2.volume();
cout << "Volume of Box2 : " << v <<endl;
return 0;
}
```

2) ***By default, all members of a class are private.***
3) ***If a member of a class is private, you cannot access it outside of the class.***

EX: Programmatically wrong example

```
#include <iostream>
using namespace std;
class Box {
    double length; // Length of a box
    double breadth; // width of a box
    double height; // Height of a box
    };
int main() {
    Box Box1; // Declare Box1 of type Box
    double volume = 0.0; // Store the volume of a box here
    // box 1 specification
```

```
Box1.height = 5.0;
Box1.length = 6.0;
Box1.breadth = 7.0;
// volume of box 1
volume = Box1.height * Box1.length * Box1.breadth;
cout << "Volume of Box1 : " << volume <<endl;
return 0;}
```



C++ has no fixed order in which you declare public and private members; you can declare them in any order. The only thing you need to remember is that, by default, all members of a class are private.[***ALWAYS USING PUBLIC FUNCTIONS TO ALTER PRIVATE DATA***]

To solve the error in the previous program we add a public function that can get the value of Box dimensions (private data)

```
#include <iostream>
using namespace std;
class Box {
```

```
double length; // Length of a box
double breadth; // width of a box
double height; // Height of a box
public:
void setDim(double l,double b,double h)
    {    length=l;
         breadth=b;
         height=h; }
void GetDim(double& l,double& b,double& h)
{       l=length;
         b=breadth;
         h=height; }};
int main() {
         Box Box1; // Declare Box1 of type Box
         double x,y,z, volume = 0.0; // Store the volume of a box here
         // box 1 specification
         Box1.setDim(5.0,6.0,7.0);
         Box1.GetDim(x,y,z);
         // volume of box 1
         volume = x*y*z;
         cout << "Volume of Box1 : " << volume <<endl;
         return 0;}
```

At this point, you might wonder, if you are going to create a public function that accesses private data, why not just make the data public in the first place, avoiding

the function? You create the public function so you, the class creator, can control how the data items are used. For example, if your radio provides an interface that allows you to set the volume only as high as 10, then you cannot set it to 11 even though the internal components of your radio might be able to produce that level of sound. Similarly, if you allow access to only public methods for a function that is a user of your class (often called a **class client**), then you control how the user can manipulate the data.