CSC 226 INTRODUCTION TO OOP
Lecture Notes 2

Basic Concepts of Object Oriented Programming

It is necessary to understand some of the concepts used extensively in object-oriented programming. These include:

- Objects
- Classes
- Data abstraction and encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message passing

We shall discuss these concepts in some detail in this section.

1 Objects

Objects are the basic run time entities in an object- oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle. Object is a real world entity, for example, chair, car, pen, mobile, laptop etc.
In other words, object is an entity that has state and behavior. Here, state means data and behavior means functionality.

They may also represent user-defined data such as vectors, time and lists. Programming problem is analyzed in term of objects and the nature of communication between them. Program objects should be chosen such that they match closely with the real-world objects. Objects take up space in the memory and have an associated address like a record in Pascal, or a structure in c.

When a program is executed, the objects interact by sending messages to one another. For example, if "customer" and "account" are to object in a program, then the customer object may send a message to the account object requesting for the bank balance. Each object contain data, and code to manipulate data. Objects can interact without having to know details of each other's data or code. It is a sufficient to know the type of message accepted, and the type of response returned by the objects. Although different author represent them differently fig 1.1 shows two notations that are popularly used in object-oriented analysis and design.

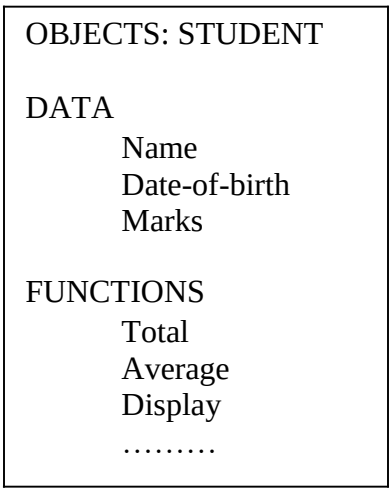

Fig. 1.1 representing an object

2 Classes

In C++, class is a group of similar objects. It is a template from which objects are created. It can have fields, methods, constructors etc.

Let's see an example of C++ class that has three fields only.

```
1.    class Student
2.     {
3.        public:
4.        int id;  //field or data member
5.        float salary; //field or data member
6.        String name;//field or data member
7.     }
```

Let's see an example of class that has two fields: id and name. It creates instance of the class, initializes the object and prints the object value.

```
1.        #include <iostream>
2.        using namespace std;
3.        class Student {
4.          public:
5.            int id;//data member (also instance variable)
6.            string name;//data member(also instance variable)
7.        };
8.        int main() {
9.           Student s1; //creating an object of Student
10.          s1.id = 002;
11.          s1.name = "Akinyemi";
12.          cout<<s1.id<<endl;
13.          cout<<s1.name<<endl;
14.          return 0;
15.       }
```

Output:

```
002
Akinyemi
```

A C++ class combines data and methods for manipulating the data into one. Classes also determine the forms of objects. The data and methods contained in a class are known as class members. We just mentioned that objects contain data, and code to manipulate that data. The entire set of data and code of an object can be made a user -defined data type with the help of class. In fact, objects are variables of the type class. Once a class has been defined, we can create any number of objects belonging to that class. Each object is associated with the data of type class with which they are created. A class is thus a collection of objects similar types. For examples, Mango, Apple and orange members of class fruit. Classes are user-defined that types and behave like the built-in types of a programming language. The syntax used to create an object is not different then the syntax used to create an integer object in C. If fruit has been defines as a class, then the statement Fruit Mango; Will create an object **mango** belonging to the class **fruit.**

C++ Classes/Objects

Everything in C++ is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an **object**. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

Attributes and methods are basically **variables** and **functions** that belongs to the class. These are often referred to as "class members".

A class is a user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects.

We can think of a class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows, etc. Based on these descriptions we build the house. House is the object.

Create a Class

To create a class, use the class keyword:

A class is defined in C++ using keyword `class` followed by the name of the class.
The body of the class is defined inside the curly brackets and terminated by a semicolon at the end.

```
class className {
   // some data
   // some functions
};
```

For example,

```
class Room {
    public:
        double length;
        double breadth;
        double height;

        double calculateArea(){
            return length * breadth;
        }

        double calculateVolume(){
            return length * breadth * height;
        }

};
```

Here, we defined a class named `Room`.
The variables `length`, `breadth`, and `height` declared inside the class are known as **data members**. And, the functions `calculateArea()` and `calculateVolume()` are known as **member functions** of a class.

Example

Create a class called "MyClass":

```
class MyClass {       // The class
  public:             // Access specifier
    int myNum;        // Attribute (int variable)
    string myString;  // Attribute (string variable)
};
```

Example explained

- The class keyword is used to create a class called MyClass.
- The public keyword is an **access specifier**, which specifies that members (attributes and methods) of the class are accessible from outside the class. You will learn more about access specifiers later.
- Inside the class, there is an integer variable myNum and a string variable myString. When variables are declared within a class, they are called **attributes**.
- At last, end the class definition with a semicolon;.

C++ Objects

When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, we need to create objects.

Syntax to Define Object in C++

```
className objectVariableName;
```

We can create objects of Room class (defined in the above example) as follows:

```
// sample function
void sampleFunction() {
    // create objects
    Room room1, room2;
}

int main(){
    // create objects
    Room room3, room4;
}
```

Here, two objects room1 and room2 of the Room class are created in sampleFunction(). Similarly, the objects room3 and room4 are created in main().
As we can see, we can create objects of a class in any function of the program. We can also create objects of a class within the class itself, or in other classes.

Also, we can create as many objects as we want from a single class.

C++ Access Data Members and Member Functions

We can access the data members and member functions of a class by using a . (dot) operator. For example,

```
room2.calculateArea();
```

This will call the calculateArea() function inside the Room class for object room2.
Similarly, the data members can be accessed as:

```
room1.length = 5.5;
```

In this case, it initializes the length variable of room1 to 5.5.

Example 1: Object and Class in C++ Programming

```
// Program to illustrate the working of
// objects and class in C++ Programming

#include <iostream>
using namespace std;

// create a class
class Room {

  public:
   double length;
   double breadth;
   double height;

   double calculateArea() {
      return length * breadth;
   }

   double calculateVolume() {
      return length * breadth * height;
   }
};

int main() {

   // create object of Room class
   Room room1;

   // assign values to data members
   room1.length = 42.5;
   room1.breadth = 30.8;
   room1.height = 19.2;

   // calculate and display the area and volume of the room
   cout << "Area of Room =  " << room1.calculateArea() << endl;
   cout << "Volume of Room =  " << room1.calculateVolume() << endl;

   return 0;
}
```

Output

```
Area of Room =  1309
Volume of Room =  25132.8
```

In this program, we have used the `Room` class and its object `room1` to calculate the area and volume of a room.
In `main()`, we assigned the values of `length`, `breadth`, and `height` with the code:

```
room1.length = 42.5;
room1.breadth = 30.8;
room1.height = 19.2;
```

We then called the functions `calculateArea()` and `calculateVolume()` to perform the necessary calculations.
Note the use of the keyword `public` in the program. This means the members are public and can be accessed anywhere from the program.
As per our needs, we can also create private members using the `private` keyword. The private members of a class can only be accessed from within the class. For example,

```
class Test {

private:
   int a;
   void function1() { }

public:
   int b;
   void function2() { }
}
```

Here, `a` and `function1()` are private. Thus they cannot be accessed from outside the class.
On the other hand, `b` and `function2()` are accessible from everywhere in the program.

Create an Object

In C++, an object is created from a class.

We have already created the class named MyClass, so now we can use this to create objects.

To create an object of MyClass, specify the class name, followed by the object name.

To access the class attributes (myNum and myString), use the dot syntax (.) on the object:

Example

Create an object called "myObj" and access the attributes:

```
class MyClass {       // The class
  public:             // Access specifier
    int myNum;        // Attribute (int variable)
    string myString;  // Attribute (string variable)
};

int main() {
  MyClass myObj;  // Create an object of MyClass

  // Access attributes and set values
  myObj.myNum = 15;
  myObj.myString = "Some text";

  // Print attribute values
  cout << myObj.myNum << "\n";
  cout << myObj.myString;
  return 0;
}
```

Multiple Objects

You can create multiple objects of one class:

Example

```
// Create a Car class with some attributes
class Car {
  public:
```

```
  string brand;
  string model;
  int year;
};

int main() {
  // Create an object of Car
  Car carObj1;
  carObj1.brand = "BMW";
  carObj1.model = "X5";
  carObj1.year = 1999;

  // Create another object of Car
  Car carObj2;
  carObj2.brand = "Ford";
  carObj2.model = "Mustang";
  carObj2.year = 1969;

  // Print attribute values
  cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";
  cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";
  return 0;
}
```