

---

## **Lab No.1 Object Programming Essentials [Part 1]**

---

### **1.1 Introduction**

In Computer Programming course, procedural programming has been covered. It lets you design a program by assembling sequence of instructions. This is the way in which traditional programming works, but now there is clear paradigm shift towards object based approach. Object-Oriented programming lets you model your program in a way in which objects, their relationships and the functions that they operate are clearly visible to the programmer. The advantage is that it gives an extended modular concept in which the interface is made public while the implementation details are kept hidden. This lets the program to grow in size without becoming too cumbersome. While in procedural programming, when a program grew beyond a certain size it became almost impossible to understand and extend especially when the programmer had forgotten minor concepts constructing the program. It must be kept in mind and acknowledged that programs are reusable entities; they have a life beyond what is normally expected. Hence designing a program that can be easily extended, modified, and interfaced with other systems are very important characteristics of any well written program. This lab has been designed to give in-depth knowledge how to make classes, objects, and their interactions.

### **1.2 Objectives of the lab**

- 1 Clearly understand the purpose and advantages of OOP
- 2 Understand the concept of a Class and Objects
- 3 Develop a basic class containing Data Members and Member Functions
- 4 Use access specifiers to access Class Members
- 5 Make Simple and Overloaded Constructor
- 6 Use the Class Objects and Member Functions to provide and extract data from Object
- 7 Practice with Classes and Objects

### **1.3 Pre-Lab**

#### **1.3.1 Class**

- 1 A set of homogeneous objects
- 2 An Abstract Data Type (ADT) -- describes a new data type
- 3 A collection of data items or functions or both

- 4 Represents an entity having
  - ☞ Characteristics
    - Attributes (member data)
  - ☞ Behavior
    - Functionality (member functions)
- 5 Provides a plan/template/blueprint
  - ☞ is just a declaration
  - ☞ is a description of a number of similar objects
- 6 Class Definition is a Type Declaration -- No Space is Allocated

### 1.3.2 Syntax of Class in C++

```
class Class_Name {
private:                                //Default and Optional
    member_specification_1;

    member_specification_n;
public:
    member_specification_n+1;

    member_specification_n+m;
};                                     //Do not forget the semicolon after the closing brace.
```

### 1.3.3 Structure of a class

- 1 Data members and member functions in a class
- 2 Data members are variables of either fundamental data types or user defined data types.
- 3 Member functions provide the interface to the data members and other procedures and function
- 4 Member access specifiers-- **public, private, and protected** used to specify the access rights to the members
  - **Private**: available to member functions of the class
  - **Protected**: available to member functions and derived classes
  - **Public**: available to entire world
- 5 The default member accessibility is **private**, meaning that only class members can access the data member or member function.

### 1.3.4 A simple program using a class in C++: complexSimple.cpp

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

class complex {
private:
    double re, im;
public:
    void set_val(double r, double i) {
        re=r;
```

```

        im=i;
    }
    void show(){
        cout<<"complex number: "<<re<<"+"<<im<<"i"<<endl;
    }
};

int main(){
    complex    c1;
    c1.set_val(4,3);
    c1.show();
    return 0;
}

```

### Output:

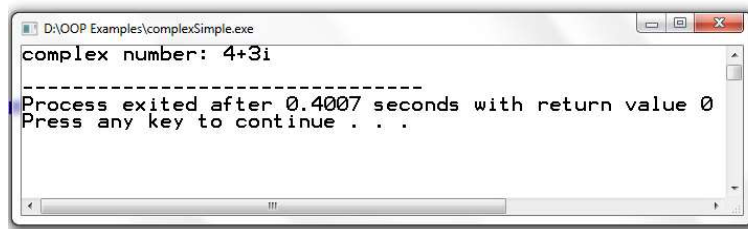


Figure 1.1: Output complexSimple.cpp

### 1.3.5 Constructor

- ☞ Special member function -- same name as the class name
  - initialization of data members
  - acquisition of resources
  - opening files
- ☞ Does NOT return a value
- ☞ Implicitly invoked when objects are created
- ☞ Can have arguments
- ☞ Cannot be explicitly called
- ☞ Multiple constructors are allowed
- ☞ If no constructor is defined in a program, default constructor is called
  - no arguments
  - constructors can be overloaded (two constructors with same name but having different signatures)

### 1.3.6 A simple program demonstrating use of constructor: complexCtr.cpp

```

#include <iostream>
using namespace std;

class complex {

```

```

private:
    double re, im;
public:
    complex(){                                // simple method
        re=0;
        im=0;
    }
    complex(double r, double i):re(r), im(i) //initializer list
    {}

    void set_val(double r, double i){
        re=r;
        im=i;
    }
    void show(){
        cout<<"complex number: "<<re<<"+ "<<im<<"i"<<endl;
    }
};

int main(){
    complex    c1(3, 4.3);
    complex    c2;
    c1.show();
    c2.set_val(2, 5.5);
    c2.show();
    return 0;
}

```

### Output:



Figure 1.2: Output complexCtr.cpp

## 1.3.7 Syntax of Class in Java

```

public class class_name {
    data_member_1;
    data_member_2;
    public type function_name();
}

```

## 1.3.8 A simple program using class in Java: complex.java

```

package lesson;
public class complex {

```

```

    double re,im;
    public complex(){
        re=0;
        im=0;
    }
    public complex(double r,double i){
        re=r;
        im=i;
    }
    public void set_val(double rr,double ii){
        re=rr;
        im=ii;
    }
    public void show(){
        System.out.println("complex number: "+re+" "+im+"i");
    }
}

```

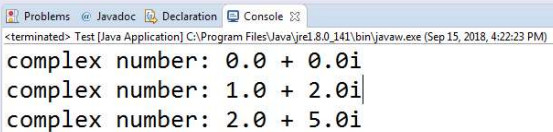
### test.java

```

package lesson;
public class Test {
    public static void main(String[] args) {
        complex obj1=new complex();
        obj1.show();
        complex obj2=new complex(1,2);
        obj2.show();
        obj1.set_val(2,5);
        obj1.show();
    }
}

```

### Output:



```

complex number: 0.0 + 0.0i
complex number: 1.0 + 2.0i
complex number: 2.0 + 5.0i

```

Figure 1.3: Output complex.java and test.java

## 1.3.9 Syntax of Class in Python

```

Class class_name:
    def __init__(self):
        //body
    def __init__(self,arg1,arg2,...):
        //body
    def function1(self):
        //body

```

### 1.3.10 A simple program using a class in Python: complex.py

```
class Complex:
    def __init__(self):
        self.re=0
        self.im=0
    def __init__(self,r,i):
        self.re=r
        self.im=i
    def set_value(self,rr,ii):
        self.re=rr
        self.im=ii
    def show(self):
        print "complex number: ",self.re,"+",self.im,"i"

obj1=Complex(1,2)
obj1.show()

obj2=Complex(0,0)
obj2.set_value(2,4)
obj2.show()
```

#### Output:

```
Python Interpreter
*** Python 2.7.10 (default, May 23 2015, 09:44:00) [MSC v.1500 64 bit (AMD64)] on win32. ***
*** Remote Python engine is active ***
>>>
*** Remote Interpreter Reinitialized ***
>>>
complex number: 1 + 2 i
complex number: 2 + 4 i
>>>
```

Figure 1.4: Output complex.py

## 1.4 Activities

**Complete all activities in C++. Python is encouraged. Java is not required.**

### 1.4.1 Activity

Create a class named **LightBulb** that represents a light bulb. The class should have the following characteristics:

1. It should contain a single Boolean field, **isOn**, which indicates whether the light bulb is on or off.
2. Define a constructor that takes no parameters. The **isOn** field should be set to false in the constructor, indicating that the bulb is initially off.
3. Define two mutator methods:
  - a. **turnOn ()**: This method should set the **isOn** field to true, turning the light bulb on.

- b. **turnOff ()**: This method should set the **isOn** field to false, turning the light bulb off.
4. Define an accessor method named **getState ()** that returns the current state of the light bulb (whether it's on or off). Make the function *const*.
5. Write a **show ()** that calls the **getState ()** to check the state of the bulb and displays the corresponding message. If the state is on, display message *"The light bulb is now ON"*. Otherwise, display message *"The light bulb is now OFF"*. Make the function *const*.
6. Demonstrate the use of the **LightBulb** class by creating an instance of the class in **main ()** function, turning the bulb on and off using the mutator methods, and displaying its state using the accessor method.

### 1.4.2 Activity

Create a class called **GPSCoordinates** that represents latitude and longitude coordinates for a location. The class should have the following characteristics:

1. The class contains two data members: **latitude** and **longitude**, which represent the latitude and longitude coordinates of a location, respectively.
2. Define two constructors:
  - a. A no-argument constructor that initializes both **latitude** and **longitude** to 0.
  - b. A two-argument constructor that takes values for **latitude** and **longitude** as parameters and sets the corresponding data members accordingly.
3. Provide separate getter and setter methods for each of the data members (**getLatitude**, **getLongitude**, **setLatitude**, **setLongitude**). The getter functions should return the corresponding values, while the setter functions should modify the coordinates.
4. Provide a **display** method that displays the coordinates in the format "(latitude, longitude)".
5. Make appropriate functions *const*.
6. Demonstrate the use of the **GPSCoordinates** class by creating instances of the class in **main ()** function.

## 1.5 Testing

### Test Cases for Activity 1.4.1

Sample Inputs	Sample Outputs
For the LightBulb Object, <b>bulb</b> :	
Call <b>show ()</b> function to view current bulb state	The light bulb is now OFF.
Call <b>turnOn ()</b> function to turn the bulb on	
Call <b>show ()</b> function to view the updated state	The light bulb is now ON.
Call <b>turnOff ()</b> function to Turn the bulb off	
Call <b>show ()</b> function to view the updated state	The light bulb is now OFF.

#### Test Cases for Activity 1.4.2

Sample Inputs	Sample Outputs
GPSCoordinates <b>location1</b> : Call <b>display</b> () function to show initial value of location1 Then set latitude to 33.6844 and longitude to 71.0975 Call <b>display</b> () function to show updated value of location1	Location 1: (0, 0)  Updated Location 1: (33.6844, 71.0975)
GPSCoordinates <b>location2</b> : Use the two-argument constructor and pass 33.6844, 73.0479 Call <b>display</b> () function to show initial value of location2	Location 2: (33.6844, 73.0479)

## 1.6 Tools

- 1 Code::Blocks for C++ (<http://www.codeblocks.org/downloads/26>)
- 2 Eclipse EE Photon for Java (<https://www.eclipse.org/downloads/>)
- 3 Visual Studio Code for Python (<https://code.visualstudio.com/download>)

## 1.7 References

- 1 Class notes
- 2 Object-Oriented Programming in C++ by *Robert Lafore* (Chapter 6)
- 3 How to Program C++ by *Deitel & Deitel* (Chapter 6)
- 4 Programming and Problem Solving with Java by *Nell Dale & Chip Weems*
- 5 Murach's Python Programming by *Micheal Urban & Joel Murach*