

---

## **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:

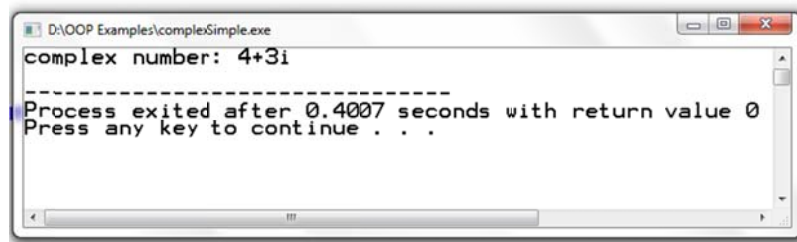


Figure 1.1: Output complexSimple.cpp

### 1.3.5 Constructor

- 1 Special member function -- same name as the class name
  - ☐ initialization of data members
  - ☐ acquisition of resources
  - ☐ opening files
- 2 Does NOT return a value
- 3 Implicitly invoked when objects are created
- 4 Can have arguments
- 6 Cannot be explicitly called
- 7 Multiple constructors are allowed
- 8 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:

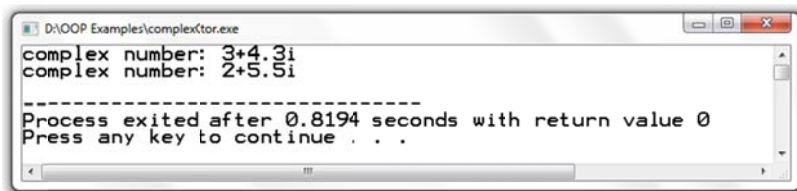


Figure 1.2: Output complexCtr.cpp

### 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");
    }
}

```

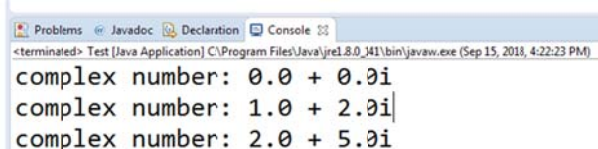
#### test.java

```

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
```

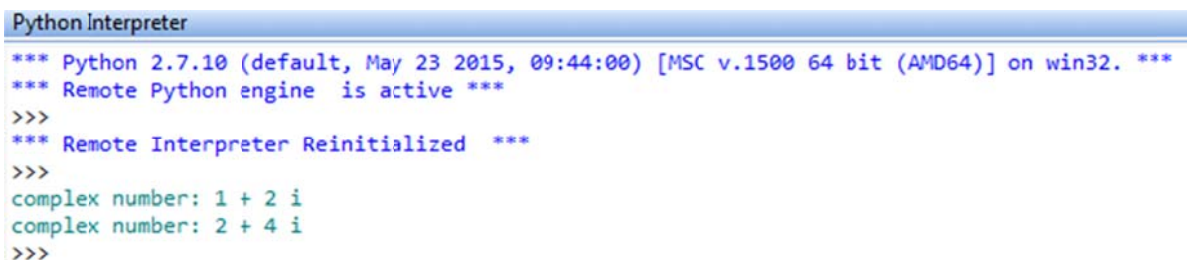
### 1.3.10 A simple program using a class in Python: complex.py

```
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

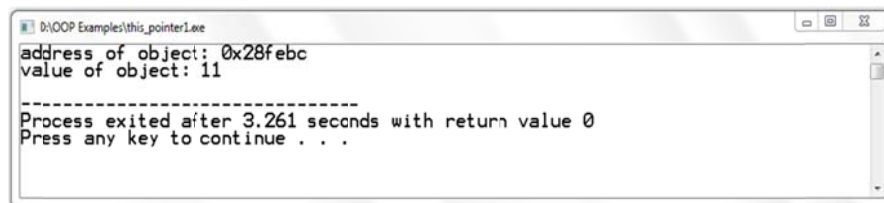
### 1.3.11 Behavior of this pointer in OOP

- 1 In C++, the member functions of every object have access to a sort of magic pointer named **this**, which points to the object itself. Thus any member function can find out the

address of the object of which it is a member. The sample program demonstrates its use. For further details, visit page number 547-550 of reference #2.

```
1  #include <iostream>
2  using namespace std;
3
4  class this_usage {
5  private:
6      int a;
7  public:
8      void tester(){
9          this -> a = 11;  // same as a = 11;
10         cout << "address of object: " << this << endl;
11         cout << "value of object: " << this -> a << endl;
12     }
13 };
14
15 int main(){
16     this_usage t;
17     t.tester();
18
19     return 0;
20 }
```

**Output:**



```
D:\OOP Examples\this_pointer1.exe
address of object: 0x28febc
value of object: 11

-----
Process exited after 3.261 seconds with return value 0
Press any key to continue . . .
```

**Figure 1.5: Output this\_pointer1.cpp**

2 In Java, keyword **this** is a reference variable that refers to the current object. Its various usages are as follows:

- It can be used to refer instance variable of current class
- It can be used to invoke or initiate current class constructor
- It can be passed as an argument in the method call
- It can be passed as argument in the constructor call
- It can be used to return the current class instance

For further details, visit <https://www.geeksforgeeks.org/this-reference-in-java/> and <https://www.guru99.com/java-this-keyword.html>.

3 The equivalent of **this** keyword in Python is **self**. It is seen in method definitions and in variable initialization. For more details, visit <https://medium.com/quick-code/understanding-self-in-python-a3704319e5f0>.

## 1.4 Activities

**Perform all these activities in C++, Python, & Java.**

### 1.4.1 Activity

Create a class, **Heater** that contains a single integer field, **temperature**. Define a constructor that takes no parameters. The **temperature** field should be set to the value 15 in the constructor. Define the mutators: **warmer** and **cooler**, whose effect is to increase or decrease the value of the temperature by 5 respectively. Define an accessor method to return the value of **temperature**. Demonstrate the use of **Heater** class.

### 1.4.2 Activity

Create a class called **Point** that has two data members: **x**- and **y**-coordinates of the point. Provide a no-argument and a 2-argument constructor. Provide separate get and set functions for the each of the data members i.e. **getX**, **getY**, **setX**, **setY**. The getter functions should return the corresponding values to the calling function. Provide a **display** method to display the point in (x, y) format. Make appropriate functions **const**.

### 1.4.3 Activity

Create a class called **BankAccount** that models a checking account at a bank. The program creates an account with an opening balance, displays the balance, makes a deposit and a withdrawal, and then displays the new balance. Note in withdrawal function, if balance is below Rs. 500 then display message showing insufficient balance otherwise allow withdrawal.

## 1.5 Testing

### Test Cases for Activity 1.4.1

Sample Inputs	Sample Outputs
For Heater Object, h1: Call show() function to view current heater value Call warmer() function to increase temperature Call show() function to view current heater value Call cooler() function to decrease temperature Call show() function to view current heater value	Temperature = 15  Temperature = 20  Temperature = 15
For Heater Object, h2: Make your own sequence consisting of show(), warmer(), and cooler() functions.	Desired Temperature

### Test Cases for Activity 1.4.2



Sample Inputs	Sample Outputs
Point p1 Call display() function to show initial value of p1 Then set x to 2 and y to 3 Call display() function to show updated value of p1	Original p1 = (0,0)  Updated p1 = (2,3)
Point p2 x: 5 y: 2 Call display() function to show p2 Then set x to 6 and y to 3 Call display() function to show updated value of p2	Original p2 = (5,2)  Updated p2 = (6,3)

#### Test Cases for Activity 1.4.3

Sample Inputs	Sample Outputs
Open Bank Account having Rs. 1000. Display current balance. Deposit Rs. 500 and display current balance Withdraw Rs. 500 and display current balance Withdraw Rs. 500 and display current balance Withdraw Rs. 500 and display current balance Display current balance.	Current Balance: 1000 Current Balance: 1500 Current Balance: 1000 Current Balance: 500 Insufficient Balance. Current Balance: 500

## 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 PyScripter for Python (<https://sourceforge.net/projects/pyscripter/>)

## 1.7 References

- 1 Class notes
- 2 Object-Oriented Programming in C++ by *Robert Lafore* (Chapter 6 and Chapter 11)
- 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*