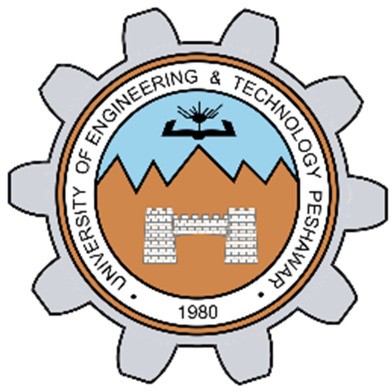
**Object Programming Essentials 2**

## LAB # 02

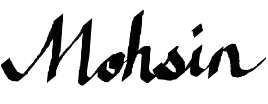


**Fall 2020**

**CSE208L Object Oriented Programming Lab**

Submitted by: **Syed Mohsin Shah** Registration No. : **19PWCSE1749** Class Section: **A**

“On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”



Student Signature:

Submitted to:

## Engr. Sumayyea Salahuddin

October 9, 2019

Department of Computer Systems Engineering University of Engineering and Technology, Peshawar

# Objectives of the Lab:

* Understand how class object can be passed and returned from class member function
* Write a class with member function having objects as arguments
* Write a class with member function that return object
* Test member function effectively using given test cases

**Activity # 01**

**Title:**

Perform arithmetic with complex numbers..

# Problem analysis:

Reuse Complex class given in section 2.3.2 (C++), section 2.3.5 (Java), and section 2.3.6 (Python) to perform arithmetic with complex numbers. Note that addition and printing is already done in given sections. Add the following public methods to perform complex subtraction and multiplication as well: Input, Subtract and Multiply.

# Algorithm:

* Create Class
* Create data members
* Create member functions
* Create objects
* Test functions by calling methods

**In C++**

**Source code:**

#include<iostream>

using namespace std;

class complex {

private:

double re, im;

public:

complex();

complex(double r, double i);

void show();

void input();

void addCom(complex c1, complex c2);

void subCom(complex c1, complex c2);

void mulCom(complex c1, complex c2);

};

complex::complex():re(0),im(0){}

complex::complex(double r, double i):re(r),im(i){}

void complex::show() {

if(im>0)

cout << re << " + " << im << "i" << endl;

else

cout << re << im << "i" << endl;

}

void complex::input() {

cout<<"Enter real number: ";

cin>>re;

cout<<"Enter imaginary number: ";

cin>>im;

}

void complex::addCom(complex c1, complex c2) {

re = c1.re + c2.re;

im = c1.im + c2.im;

}

void complex::subCom(complex c1, complex c2) {

re = c1.re - c2.re;

im = c1.im - c2.im;

}

void complex::mulCom(complex c1, complex c2) {

re = (c1.re \* c2.re) - (c1.im \* c2.im);

im = (c1.re \* c2.im) + (c2.re \* c1.im);

}

main() {

complex c1,c2,c;

c1.input();

cout<<"C1 = ";

c1.show();

c2.input();

cout<<"C2 = ";

c2.show();

c.addCom(c1, c2);

cout<<"Sum = ";

c.show();

c.subCom(c1, c2);

cout<<"Difference = ";

c.show();

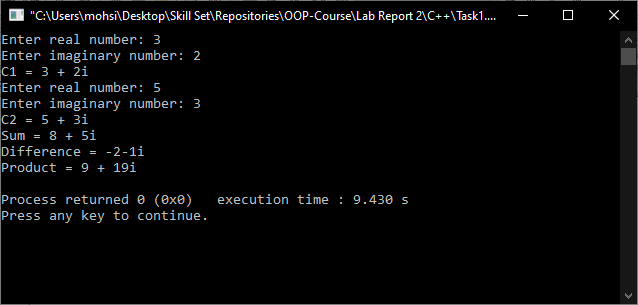
c.mulCom(c1, c2);

cout<<"Product = ";

c.show();

}

**Output:**

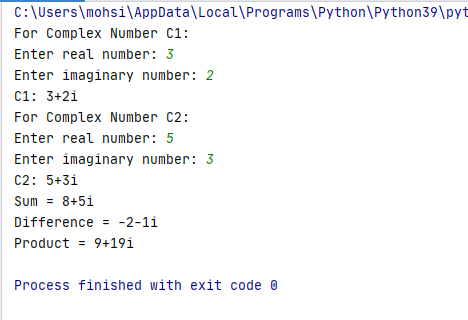


**In Python**

**Source code:**

class Complex:  
 re = 0  
 im = 0  
  
 def \_\_init\_\_(self):  
 self.re = 0  
 self.im = 0  
  
 def \_\_init\_\_(self, r, i):  
 self.re = r  
 self.im = i  
  
 def show(self):  
 if self.im >= 0:  
 print(**f"**{self.re}**+**{self.im}**i"**)  
 else:  
 print(self.re, self.im, **"i"**, sep=**""**)  
  
 def addCom(self, c1, c2):  
 self.re = c1.re + c2.re  
 self.im = c1.im + c2.im  
  
 def subCom(self, c1, c2):  
 self.re = c1.re - c2.re  
 self.im = c1.im - c2.im  
  
 def mulCom(self, c1, c2):  
 self.re = (c1.re \* c2.re) - (c1.im \* c2.im)  
 self.im = (c1.re \* c2.im) + (c2.re \* c1.im)  
  
 def input(self):  
 self.re = int(input(**"Enter real number: "**))  
 self.im = int(input(**"Enter imaginary number: "**))  
  
  
c1 = Complex(0,0)  
c2 = Complex(0,0)  
c = Complex(0,0)  
  
print(**"For Complex Number C1:"**)  
c1.input()  
print(**"C1:"**, end = **" "**)  
c1.show()  
  
print(**"For Complex Number C2:"**)  
c2.input()  
print(**"C2:"**, end = **" "**)  
c2.show()  
  
print(**"Sum ="**, end = **" "**)  
c.addCom(c1, c2)  
c.show()  
  
print(**"Difference ="**, end = **" "**)  
c.subCom(c1, c2)  
c.show()  
  
print(**"Product ="**, end = **" "**)  
c.mulCom(c1, c2)  
c.show()

**Output:**

****

**In Java (Optional)**

**Source code:**

**Complex.java:**

**package** task1;

**import** java.util.\*;

**public** **class** Complex {

**private** **double** re, im;

**private** Scanner s1;

**public** Complex () {

**this**.re = 0;

**this**.im = 0;

}

**public** Complex(**double** r, **double** i) {

**this**.re = r;

**this**.im = i;

}

**public** **void** Input() {

s1 = **new** Scanner(System.***in***);

System.***out***.print("Real Value = ");

**this**.re = s1.nextDouble();

System.***out***.print("Imaginary Value = ");

**this**.im = s1.nextDouble();

}

**public** **void** Show() {

**if** (**this**.im >= 0) {

System.***out***.println(**this**.re + "+" + **this**.im + "i");

} **else** {

System.***out***.println(**this**.re + **this**.im + "i");

}

}

**public** **void** AddCom(Complex c1, Complex c2) {

**this**.re = c1.re + c2.re;

**this**.im = c1.im + c2.im;

}

**public** **void** SubCom(Complex c1, Complex c2) {

**this**.re = c1.re - c2.re;

**this**.im = c1.im - c2.im;

}

**public** **void** MulCom(Complex c1, Complex c2) {

**this**.re = (c1.re \* c2.re) - (c1.im \* c2.im);

**this**.im = (c1.re \* c2.im) + (c2.re \* c1.im);

}

}

**Main.java:**

**package** task1;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Complex c1 = **new** Complex();

Complex c2 = **new** Complex();

Complex c = **new** Complex();

System.***out***.println("Write values for C1: ");

c1.Input();

System.***out***.print("C1: ");

c1.Show();

System.***out***.println("\nWrite values for C2: ");

c2.Input();

System.***out***.print("C2: ");

c2.Show();

System.***out***.print("\nSum = ");

c.AddCom(c1, c2);

c.Show();

System.***out***.print("\nDifference = ");

c.SubCom(c1, c2);

c.Show();

System.***out***.print("\nProduct = ");

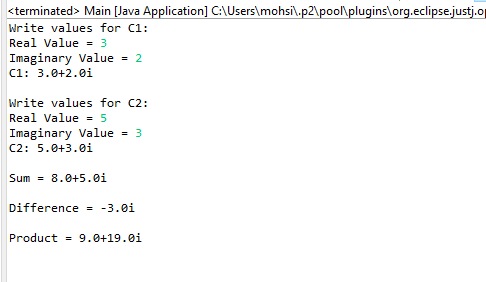
c.MulCom(c1, c2);

c.Show();

}

}

**Output:**

****

**Conclusion:**

This program helps us in understanding the basic concepts of classes and objects in different languages. It acts as a base for us and helps us in preparing ourselves for the higher level of programming. We get to know about the constructor and method in OOP with the help of this program.

**Activity # 02**

**Title:**

Return class as well.

# Problem analysis:

# Reuse Complex class written in Activity 2.4.1 to modify the addCom(), subCom(), and mulCom() class

# functions. Instead of passing two objects in each, now pass only one object. Change the return type of

# each function to complex. Adjust the function code to match the changes. Demonstrate and test the

# modified class and its objects using test cases given in Section 2.5.

# Algorithm:

* Create Class
* Create data members
* Create member functions
* Create objects
* Test functions by calling methods

**In C++**

**Source code:**

#include<iostream>

using namespace std;

class complex {

private:

double re, im;

public:

complex();

complex(double r, double i);

// complex negate(); //returns complex object

void show();

void input();

complex addCom(complex c);

complex subCom(complex c);

complex mulCom(complex c);

};

complex::complex():re(0),im(0){}

complex::complex(double r, double i):re(r),im(i){}

void complex::show() {

if(im>0)

cout << re << "+" << im << "i" << endl;

else

cout << re << im << "i" << endl;

}

void complex::input() {

cout<<"Enter real number: ";

cin>>re;

cout<<"Enter imaginary number: ";

cin>>im;

}

complex complex::addCom(complex c) {

complex x;

x.re = re + c.re;

x.im = im + c.im;

return x;

}

complex complex::subCom(complex c) {

complex x;

x.re = re - c.re;

x.im = im - c.im;

return x;

}

complex complex::mulCom(complex c) {

complex x;

x.re = (re \* c.re) - (im \* c.im);

x.im = (re \* c.im) + (c.re \* im);

return x;

}

main() {

complex c1,c2,c;

c1.input();

cout<<"C1 = ";

c1.show();

c2.input();

cout<<"C2 = ";

c2.show();

c = c1.addCom(c2);

cout<<"Sum = ";

c.show();

c = c1.subCom(c2);

cout<<"Difference = ";

c.show();

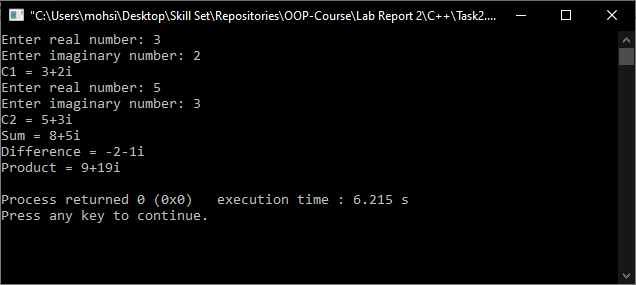
c = c1.mulCom(c2);

cout<<"Product = ";

c.show();

}

**Output:**

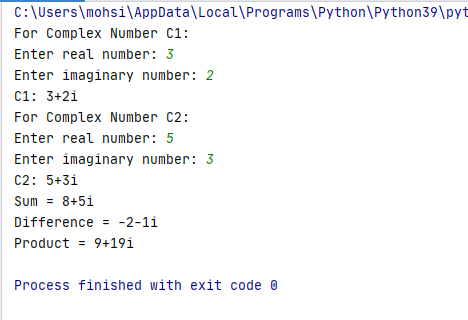


**In Python**

**Source code:**

class Point:  
class Complex:  
 re = 0  
 im = 0  
  
 def \_\_init\_\_(self):  
 self.re = 0  
 self.im = 0  
  
 def \_\_init\_\_(self, r, i):  
 self.re = r  
 self.im = i  
  
 def show(self):  
 if self.im >= 0:  
 print(**f"**{self.re}**+**{self.im}**i"**)  
 else:  
 print(self.re, self.im, **"i"**, sep=**""**)  
  
 def addCom(self, c1):  
 c = Complex(0,0)  
 c.re = self.re + c1.re  
 c.im = self.im + c1.im  
 return c  
  
 def subCom(self, c1):  
 c = Complex(0,0)  
 c.re = self.re - c1.re  
 c.im = self.im - c1.im  
 return c  
  
 def mulCom(self, c1):  
 c = Complex(0,0)  
 c.re = (c1.re \* self.re) - (c1.im \* self.im)  
 c.im = (c1.re \* self.im) + (self.re \* c1.im)  
 return c  
  
 def input(self):  
 self.re = int(input(**"Enter real number: "**))  
 self.im = int(input(**"Enter imaginary number: "**))  
  
  
c1 = Complex(0,0)  
c2 = Complex(0,0)  
c = Complex(0,0)  
  
print(**"For Complex Number C1:"**)  
c1.input()  
print(**"C1:"**, end = **" "**)  
c1.show()  
  
print(**"For Complex Number C2:"**)  
c2.input()  
print(**"C2:"**, end = **" "**)  
c2.show()  
  
print(**"Sum ="**, end = **" "**)  
c = c1.addCom(c2)  
c.show()  
  
print(**"Difference ="**, end = **" "**)  
c = c1.subCom(c2)  
c.show()  
  
print(**"Product ="**, end = **" "**)  
c = c1.mulCom(c2)  
c.show()

**Output:**

****

**In Java (Optional)**

**Source code:**

**Complex.java:**

**package** task2;

**import** java.util.\*;

**public** **class** Complex {

**private** **double** re, im;

**public** Complex () {

**this**.re = 0;

**this**.im = 0;

}

**public** Complex(**double** r, **double** i) {

**this**.re = r;

**this**.im = i;

}

**public** **void** Input() {

System.***out***.print("Real Value = ");

Scanner input = **new** Scanner(System.***in***);

**this**.re = input.nextDouble();

System.***out***.print("Imaginary Value = ");

**this**.im = input.nextDouble();

}

**public** **void** Show() {

**if** (**this**.im >= 0) {

System.***out***.println(**this**.re + "+" + **this**.im + "i");

} **else** {

System.***out***.println(**this**.re + **this**.im + "i");

}

}

**public** Complex AddCom(Complex c1) {

Complex c = **new** Complex();

c.re = **this**.re + c1.re;

c.im = **this**.im + c1.im;

**return** c;

}

**public** Complex SubCom(Complex c1) {

Complex c = **new** Complex();

c.re = **this**.re - c1.re;

c.im = **this**.im - c1.im;

**return** c;

}

**public** Complex MulCom(Complex c1) {

Complex c = **new** Complex();

c.re = (c1.re \* **this**.re) - (c1.im \* **this**.im);

c.im = (c1.re \* **this**.im) + (**this**.re \* c1.im);

**return** c;

}

}

**Main.java:**

**package** task2;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Complex c1 = **new** Complex();

Complex c2 = **new** Complex();

Complex c = **new** Complex();

System.***out***.println("Write values for C1: ");

c1.Input();

System.***out***.print("C1: ");

c1.Show();

System.***out***.println("\nWrite values for C2: ");

c2.Input();

System.***out***.print("C2: ");

c2.Show();

System.***out***.print("\nSum = ");

c = c1.AddCom(c2);

c.Show();

System.***out***.print("\nDifference = ");

c = c1.SubCom(c2);

c.Show();

System.***out***.print("\nProduct = ");

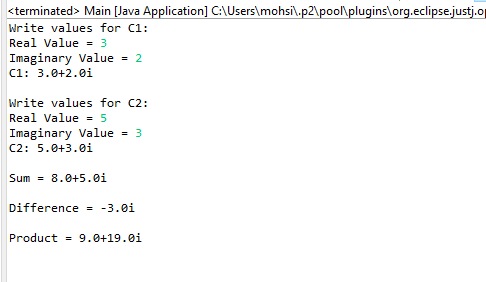
c = c1.MulCom(c2);

c.Show();

}

}

**Output:**

****

**Conclusion:**

By performing this task, it clarifies the basics of Classes and objects and function calling and making use of Non-Parameterized constructors and Constructors with Parameters.

**Activity # 03**

**Title:**

Make class of IntegerSets.

# Problem analysis:

# Create a class called IntegerSet. Each object of class IntegerSet can hold integers in the range 0 through 49. A set is represented internally as an array of ones and zeros. Array element a[ i ] is 1 if integer i is in the set. Array element a[ j ] is 0 if integer j is not in the set. The default constructor initializes a set to

# empty set i.e., a set whose array representation contains all zeros. Do Union, Intersect, IsEqualTo, Insert and Delete.

# Algorithm:

* .
* Create Class
* Create data members
* Create member functions
* Create objects
* Test functions by calling methods

**In C++**

**Source code:**

#include<iostream>

using namespace std;

class IntegerSet {

private:

int set[50];

public:

IntegerSet();

void NewIntegerSet(int arr[]);

IntegerSet UnionOfIntegerSets(IntegerSet i);

IntegerSet IntersectionOfIntegerSets(IntegerSet i);

void InsertElement(int k);

void DeleteElement(int m);

void SetPrint();

bool IsEqualTo(IntegerSet i);

};

IntegerSet::IntegerSet() {

for(int i = 0; i < 50; i++) {

set[i] = 0;

}

}

void IntegerSet::NewIntegerSet(int arr[]){

for(int i = 0; i<50; i++) {

set[i] = arr[i];

}

}

IntegerSet IntegerSet::UnionOfIntegerSets(IntegerSet i) {

IntegerSet j;

for (int n = 0; n < 50; n++) {

j.set[n] = set[n] && i.set[n];

}

return j;

}

IntegerSet IntegerSet::IntersectionOfIntegerSets(IntegerSet i) {

IntegerSet j;

for (int n = 0; n < 50; n++) {

j.set[n] = set[n] || i.set[n];

}

return j;

}

void IntegerSet::InsertElement(int k) {

set[k] = 1;

}

void IntegerSet::DeleteElement(int m) {

set[m] = 0;

}

void IntegerSet::SetPrint(){

for (int i = 0; i < 50; i++) {

cout<<set[i]<<" ";

}

cout<<endl;

}

bool IntegerSet::IsEqualTo(IntegerSet i){

for (int n = 0; n < 50; n++) {

if(set[n] != i.set[n]){

return false;

}

return true;

}

}

main() {

int x[50],y[50];

cout<<"Set values of Array X: ";

for (int i = 0; i < 50; i++) {

cin>>x[i];

if(x[i] != 0) x[i] = 1;

}

cout<<"Set values of Array Y: ";

for (int i = 0; i < 50; i++) {

cin>>y[i];

if(y[i] != 0) y[i] = 1;

}

IntegerSet i1,i2,i3;

i1.NewIntegerSet(x);

cout<<"I1: ";

i1.SetPrint();

i2.NewIntegerSet(y);

cout<<"I2: ";

i2.SetPrint();

int k,m;

cout<<"\nType location of insertion in I1: ";

cin>>k;

i1.InsertElement(k);

cout<<"Type location of deletion in I1: ";

cin>>m;

i1.DeleteElement(m);

cout<<endl;

i3 = i1.UnionOfIntegerSets(i2);

i3.SetPrint();

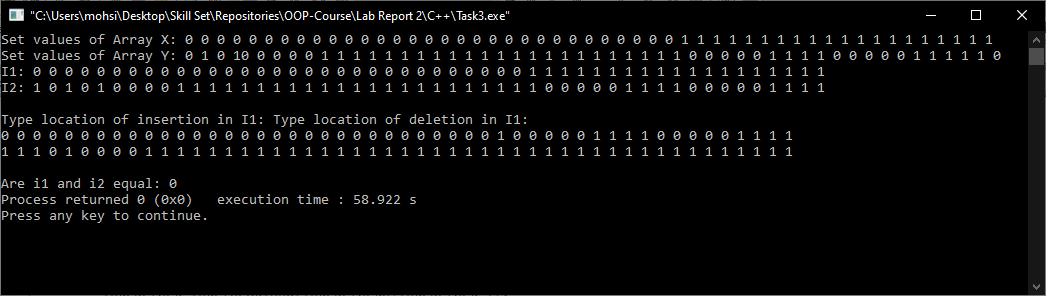
i3 = i1.IntersectionOfIntegerSets(i2);

i3.SetPrint();

cout<<"\nAre i1 and i2 equal: "<<i1.IsEqualTo(i2);

}

**Output:**

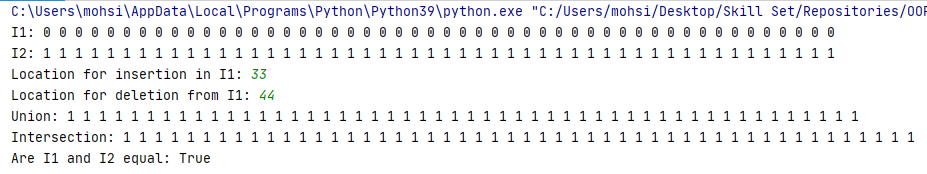


**In Python**

**Source code:**

class IntegerSet:  
 set = [0] \* 50  
  
 def \_\_init\_\_(self):  
 pass  
  
 def NewIntegerSet(self, arr):  
 for i in range(50):  
 self.set[i] = arr[i]  
  
 def UnionOfIntegerSets(self, i):  
 j = IntegerSet()  
 for n in range(50):  
 j.set[n] = self.set[n] or i.set[n]  
 return j  
  
 def IntersectionOfIntegerSets(self, i):  
 j = IntegerSet()  
 for n in range(50):  
 j.set[n] = self.set[n] and i.ser[n]  
 return j  
  
 def InsertElement(self, k):  
 self.set[k] = 1  
  
 def DeleteElement(self, m):  
 self.set[m] = 0  
  
 def SetPrint(self):  
 for i in self.set:  
 print(self.set[i], end=**" "**)  
 print()  
  
 def IsEqualTo(self, i):  
 if self.set == i.set:  
 return True  
 else:  
 return False  
  
  
*# print("Enter values for X:", end=" ")*x = [0 for i in range(50)]  
  
*# print("Enter values for Y:", end=" ")*y = [1 for i in range(50)]  
  
i1 = IntegerSet()  
i2 = IntegerSet()  
i3 = IntegerSet()  
  
i1.NewIntegerSet(x)  
print(**"I1:"**, end=**" "**)  
i1.SetPrint()  
  
i2.NewIntegerSet(y)  
print(**"I2:"**, end=**" "**)  
i2.SetPrint()  
  
k = int(input(**"Location for insertion in I1: "**))  
i1.InsertElement(k)  
  
m = int(input(**"Location for deletion from I1: "**))  
i1.DeleteElement(m)  
  
i3 = i1.UnionOfIntegerSets(i2)  
print(**"Union:"**, end=**" "**)  
i3.SetPrint()  
  
i3 = i1.UnionOfIntegerSets(i2)  
print(**"Intersection:"**, end=**" "**)  
i3.SetPrint()  
  
print(**"Are I1 and I2 equal:"**, end=**" "**)  
print(i1.IsEqualTo(i2));

**Output:**

****

**In Java (Optional)**

**Source code:**

**IntegerSet.java:**

**package** task3;

**public** **class** IntegerSet {

**private** **int** set[] = **new** **int**[50];

**public** IntegerSet() {

**for**(**int** i = 0; i < 50; i++) {

set[i] = 0;

}

}

**public** **void** NewIntegerSet(**int** arr[]) {

**for**(**int** n = 0; n < 50; n++) {

**this**.set[n] = arr[n];

}

}

**public** IntegerSet UnionOfIntegerSets(IntegerSet i) {

IntegerSet j = **new** IntegerSet();

**for**(**int** n = 0; n < 50; n++) {

**boolean** x = (**this**.set[n] != 0) || (i.set[n] != 0);

j.set[n] = x ? 1 : 0;

}

**return** j;

}

**public** IntegerSet IntersectionOfIntegerSets(IntegerSet i) {

IntegerSet j = **new** IntegerSet();

**for**(**int** n = 0; n < 50; n++) {

**boolean** x = (**this**.set[n] != 0) && (i.set[n] != 0);

j.set[n] = x ? 1 : 0;

}

**return** j;

}

**public** **void** InsertElement(**int** k) {

**this**.set[k] = 1;

}

**public** **void** DeleteElement(**int** m) {

**this**.set[m] = 0;

}

**public** **void** SetPrint() {

**for**(**int** i = 0; i < 50; i++) {

System.***out***.print(set[i] + " ");

}

System.***out***.println();

}

**public** **boolean** IsEqualTo(IntegerSet i) {

**for**(**int** n = 0; n < 50; n++) {

**if**(**this**.set[n] != i.set[n])

**return** **false**;

}

**return** **true**;

}

}

**Main.java:**

**package** task3;

**import** java.util.Scanner;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**int** x[] = **new** **int**[50], y[] = **new** **int**[50];

Scanner input = **new** Scanner(System.***in***);

System.***out***.print("Input Integer Set I1: ");

**for**(**int** i = 0; i < 50; i++) {

x[i] = input.nextInt();

}

System.***out***.print("Input Integer Set I2: ");

**for**(**int** i = 0; i < 50; i++) {

y[i] = input.nextInt();

}

IntegerSet i1 = **new** IntegerSet();

IntegerSet i2 = **new** IntegerSet();

IntegerSet i3 = **new** IntegerSet();

i1.NewIntegerSet(x);

System.***out***.print("Integer Set I1: ");

i1.SetPrint();

i2.NewIntegerSet(y);

System.***out***.print("Integer Set I2: ");

i2.SetPrint();

System.***out***.print("Insert Element in I1: ");

**int** k = input.nextInt();

i1.InsertElement(k);

System.***out***.print("I1 after inserting: ");

i1.SetPrint();

System.***out***.print("Delete Element from I1: ");

**int** m = input.nextInt();

i1.DeleteElement(m);

System.***out***.print("I1 after Deleting: ");

i1.SetPrint();

System.***out***.print("Union of I1 and I2: ");

i3 = i1.UnionOfIntegerSets(i2);

i3.SetPrint();

System.***out***.print("Intersection of I1 and I2: ");

i3 = i1.IntersectionOfIntegerSets(i2);

i3.SetPrint();

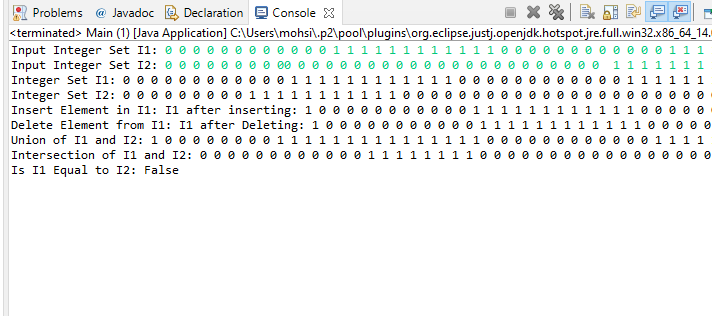
System.***out***.print("Is I1 Equal to I2: ");

System.***out***.println(i1.IsEqualTo(i2) ? "True" : "False");

}

}

**Output:**

****

**Conclusion:**

By performing this task we were able to get to know about the constructor and method in OOP. And making use of Classes and Their attributes and instances. Performing this task in various programming languages. We were able to know about the syntax and the execution mechanism.

**Registration #:**  **Name & Section:**  **Date:**

## CSE 208L – OBJECT ORIENTED PROGRAMMING LAB LAB 01 ASSESSMENT RUBRICS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Dimension** | **Exemplary** | **Acceptable** | **Developing** | **Unsatisfactory** | **Student Score out of 10 Marks** |
| **10** | **9‐7** | **6‐5** | **4‐0** |
| **Submission** | Report is submitted on time and in correct format. | Report is submitted on time with slight incorrect format. | Report is submitted on time in incorrect format. | Report is not submitted. |  |
| **Overall Impression of Lab Report** | Report is complete, well written, and organized appropriately with additional elements that enhance it. Task titles and output screenshots are included. Purpose for each concept, input requirements and output results is noted. | Report is complete, briefly written, and organized. Lacks additional elements. Task titles and output screenshots are included. Purpose for each concept, input requirements and output results is noted. | Report is mostly complete, loosely written, and fairly organized. Basic documentation including descriptions of all concepts. Specific purpose is noted for each concept. Task titles and output screenshots are  included and good formatting. | Report is incomplete, sloppy, and/or disorganized.  No documentation included.  No task titles, no output screenshots, poor  formatting. |  |
| **Ability to Code Required Class/Classes** | Able to code required class, use objects effectively, and produces desired results. | Able to code required class, use objects effectively, and produces most results. | Able to code required class, somewhat use of objects, and some results are produced. | Unable to code required class or unable to use objects. |  |
| **Compilation, Execution, and Results** | Program compiles with no errors and no warnings.  Executes without errors, excellent user prompts, good use of symbols, and spacing in output.  Thorough and organized testing has  been completed and output from test cases is included. | Program compiles with no errors and some warnings. Executes without errors.  User prompts are understandable, minimum use of symbols or spacing in output. Most of the testing has been  completed. | Program compiles with no errors and lots of warnings.  Executes without errors.  User prompts are understandable, minimum use of symbols or spacing in output.  Some testing has been  completed. | Program fails to compile. Does not execute due to errors.  User prompts are misleading or non‐ existent.  No testing has been d. |  |

**Marks**: /4 =

**Teacher Remarks and Signature:**

Department of Computer Systems Eng. UET Peshawar Prepared By: Engr. Sumayyea Salahuddin.