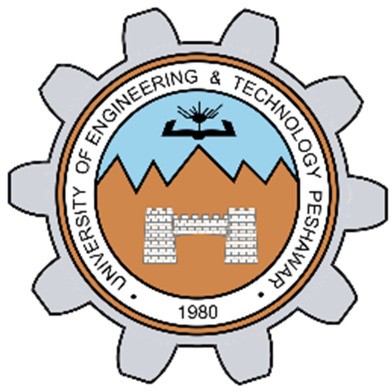
**Object Programming Essentials**

## LAB # 01

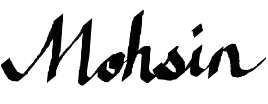


**Fall 2020**

**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:

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

**Activity # 01**

**Title:**

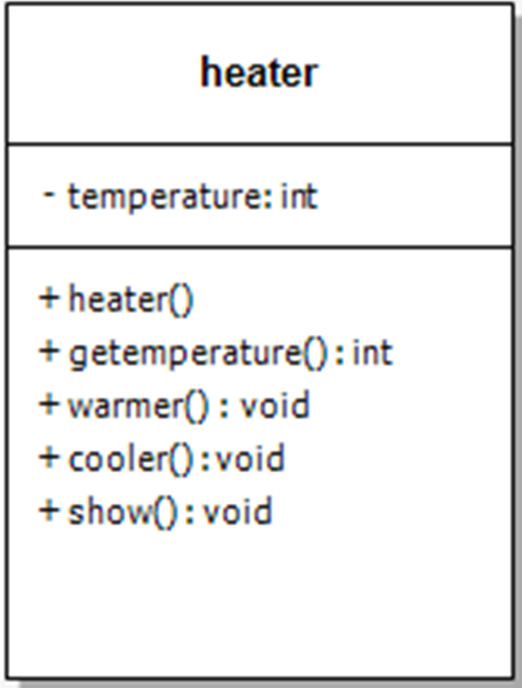
Make a class for heater and model it using temperature.

# Problem analysis:

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.

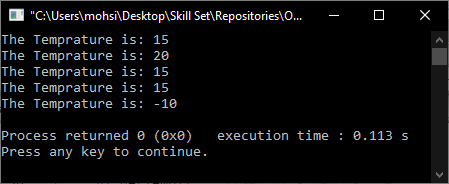
# Algorithm:

UML diagram for the above problem is given below:

* First make class heater
* Declare temperature as private integer field
* Define no argument constructor to set value of temperature to 15
* Define gettemperature method to return value of temperature
* Define warmer and cooler method to increase and decrease temperature by 5 respectively
* Define show function to display the output
* In main function, make objects of heater to demonstrate the use of heater
* Call each function one after the other and display the show function as shown in the flow chart.

**In C++**

**Source code:**

#include<iostream>

using namespace std;

class Heater

{

private:

int temprature;

public:

Heater(){

temprature = 15;

}

void warmer(){

temprature += 5;

}

void cooler(){

temprature -= 5;

}

void getTemp(){

cout<<"The Temprature is: "<<temprature<<endl;

}

};

main()

{

Heater h1; //Create h1 Heater

h1.getTemp(); //Get h1's temprature

h1.warmer(); //h1 Heater is cooled

h1.getTemp(); //get h1's temprature after cooling

h1.cooler();

h1.getTemp();

Heater h2; //Create h2 Heater

h2.getTemp(); //get h2's temprature

h2.cooler(); //h2 Heater is warmed

h2.cooler();

h2.cooler();

h2.cooler();

h2.cooler();

h2.getTemp(); //get h2 Heater's temprature after cooling

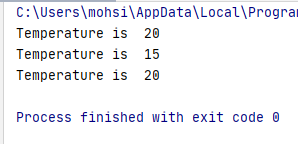
//Oh it is below freezing

}

**Output:**

**In Python**

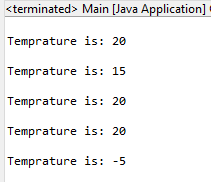
**Source code:**

****class Heater:  
 temperature = 0  
  
 def \_\_init\_\_(self):  
 self.temperature = 20  
  
 def warmer(self):  
 self.temperature += 5  
  
 def cooler(self):  
 self.temperature -= 5  
  
 def display(self):  
 print(**"Temperature is "**, self.temperature)  
  
  
h1 = Heater()  
  
h1.display()  
h1.cooler()  
h1.display()  
h1.warmer()  
h1.display()

**Output:**

**In Java (Optional)**

**Source code:**

**Heater.java:**

**package** task1;

**public** **class** Heater {

**int** temprature;

**public** Heater() {

temprature = 20;

}

**public** **void** warmer() {

temprature += 5;

}

**public** **void** cooler() {

temprature -= 5;

}

**public** **void** display() {

System.out.println("\nTemprature is: " + temprature);

}

}

**Main.java:**

**package** task1;

**class** Main {

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

Heater h1 = **new** Heater();

h1.display();

h1.cooler();

h1.display();

h1.warmer();

h1.display();

Heater h2 = **new** Heater();

h2.display();

h2.cooler();

h2.cooler();

h2.cooler();

h2.cooler();

h2.cooler();

h2.display();

}

}

**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:**

Make a class for point and model it using x and y coordinates.

# Problem analysis:

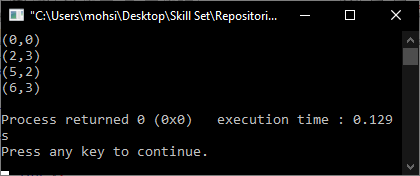
# Create a class called Point that has two data members: x‐ and y‐coordinates of the point. Provide a no‐argument and a 2‐argument constructor. Provide separate get and set functions for the each of the data members i.e. getX, getY, setX, setY. The getter functions should return the corresponding values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

# Algorithm:

* First make class point
* Declare x & y as private float field
* Define no argument constructor to set value of x & y to 0
* Define overloaded constructor to set value for x & y.
* Define getX and getY method to get x and y coordinate from user.
* Define setX and setY method to set values of x and y coordinates.
* Define setting method to set value for x & y.
* Define the show function to display the data.
* Call each function one after the other and display the show function as shown in the flow chart.

**In C++**

**Source code:**

#include<iostream>

using namespace std;

class Point

{

private:

int x;

int y;

public:

Point(){

x = 0;

y = 0;

}

Point(int x,int y){

this->x = x;

this->y = y;

}

void setX(int x){

this->x = x;

}

void setY(int y){

this->y = y;

}

void getX(){

cout<<x;

}

void getY(){

cout<<y;

}

void display(){

cout<<"("<<x<<","<<y<<")"<<endl;

}

};

main()

{

Point p1;

p1.display();

p1.setX(2);

p1.setY(3);

p1.display();

Point p2(5,2);

p2.display();

p2.setX(6);

p2.setY(3);

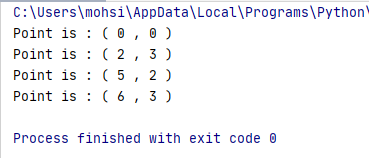
p2.display();

}

**Output:**

**In Python**

**Source code:**

****

class Point:  
 x = 0  
 y = 0  
  
 def \_\_init\_\_(self):  
 self.x = 0  
 self.y = 0  
  
 def \_\_init\_\_(self, a, b):  
 self.x = a  
 self.y = b  
  
 def getX(self):  
 print(self.x)  
  
 def getY(self):  
 print(self.y)  
  
 def setX(self, a):  
 self.x = a  
  
 def setY(self, b):  
 self.y = b  
  
 def display(self):  
 print(**"Point is : ("**, self.x, **","**, self.y, **")"**)  
  
  
p1 = Point(0,0)  
  
p1.display()  
p1.setX(2)  
p1.setY(3)  
p1.display()  
  
p2 = Point(5,2)  
  
p2.display()  
p2.setX(6)  
p2.setY(3)  
p2.display()

**Output:**

**In Java (Optional)**

**Source code:**

**Point.java:**

**package** task2;

**public** **class** Point {

**int** x;

**int** y;

**public** Point() {

x = 0;

y = 0;

}

**public** Point(**int** x, **int** y) {

**this**.x = x;

**this**.y = y;

}

**public** **void** getX() {

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

}

**public** **void** getY() {

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

}

**public** **void** setX(**int** x) {

**this**.x = x;

}

**public** **void** setY(**int** y) {

**this**.y = y;

}

**public** **void** display() {

System.***out***.println("\nPoint is: (" + x + "," + y + ")");

}

}

**Main.java:**

**package** task2;

**class** Main {

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

Point p1 = **new** Point();

p1.display();

p1.setX(2);

p1.setY(3);

p1.display();

Point p2 = **new** Point(5,2);

p2.display();

p2.setX(6);

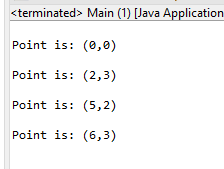
p2.setY(3);

p2.display();

}

}

**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 a class for BankAccount and model by making transactions methods.

# Problem analysis:

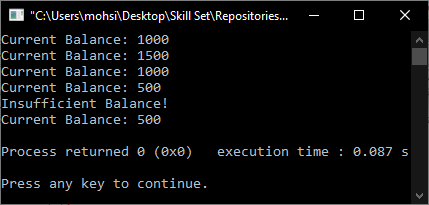
# Create a class called BankAccount that models a checking account at a bank. The program creates an account with an opening balance, displays the balance, makes a deposit and a withdrawal, and then displays the new balance. Note in withdrawal function, if balance is below Rs. 500 then display message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

# Algorithm:

* First make class BankAccount
* Declare balance as private double field
* Define no argument constructor to set value of balance to 1000.
* Define overloaded constructor to set value for balance.
* Define Deposit method to add increment the already present balance.
* Define Withdraw method to decrement the value of current balance.
* By using if/else statement in Withdraw method a pop-up message of insufficient balance is displayed.
* Define Print function to display the current balance.
* In int main with the help of variables and Do while loop the functions and values are accessed.

**In C++**

**Source code:**



#include<iostream>

using namespace std;

class BankAccount

{

private:

int balance;

public:

BankAccount(){

balance = 0;

}

BankAccount(int amount){

balance = amount;

}

void deposit(int amount){

balance += amount;

}

void withdraw(int amount){

if(balance <= 500)

cout<<"Insufficient Balance!"<<endl;

else

balance -= amount;

}

void display(){

cout<<"Current Balance: "<<balance<<endl;

}

};

main()

{

BankAccount Acc1(1000);

Acc1.display();

Acc1.deposit(500);

Acc1.display();

Acc1.withdraw(500);

Acc1.display();

Acc1.withdraw(500);

Acc1.display();

Acc1.withdraw(500);

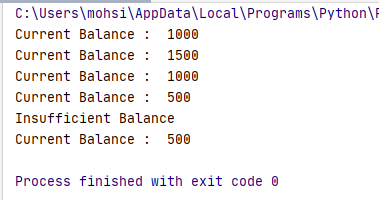
Acc1.display();

}

**Output:**

**In Python**

**Source code:**

****

class BankAccount:  
 balance = 1000  
  
 def \_\_init\_\_(self):  
 balance = 1000  
  
 def deposit(self, amount):  
 self.balance += amount  
  
 def withdraw(self, amount):  
 if self.balance <= 500:  
 print(**"Insufficient Balance"**)  
 else:  
 self.balance -= amount  
  
 def display(self):  
 print(**"Current Balance : "**, self.balance)  
  
  
Acc1 = BankAccount()  
  
Acc1.display()  
Acc1.deposit(500)  
Acc1.display()  
Acc1.withdraw(500)  
Acc1.display()  
Acc1.withdraw(500)  
Acc1.display()  
Acc1.withdraw(500)  
Acc1.display()

**Output:**

**In Java (Optional)**

**Source code:**

**BankAccount.java:**

**package** task3;

**public** **class** BankAccount {

**private** **int** balance;

**public** BankAccount() {

balance = 1000;

}

**public** **void** deposit(**int** amount) {

balance += amount;

}

**public** **void** withdraw(**int** amount) {

**if**(balance <= 500)

System.***out***.println("Insufficient Balance!");

**else**

balance -= amount;

}

**public** **void** display() {

System.***out***.println("Current Balance: " + balance);

}

}

**Main.java:**

**package** task3;

**class** Main {

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

BankAccount Acc1 = **new** BankAccount();

Acc1.display();

Acc1.deposit(500);

Acc1.display();

Acc1.withdraw(500);

Acc1.display();

Acc1.withdraw(500);

Acc1.display();

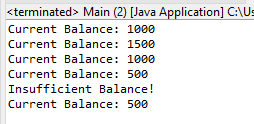
Acc1.withdraw(500);

Acc1.display();

}

}

**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.