Laboratory Activity No. 1	
Introduction to Object-Oriented Programming	
Course Code: CPE009B	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Performed:** 9/15/24
Section: CPE21S4	**Date Submitted:** 9/15/24
Name: Planta, Calvin Earl L.	**Instructor:** Dr. Sayo

1. Objective(s):

This activity aims to familiarize students with the concepts of Object-Oriented Programming

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Identify the possible attributes and methods of a given object
2.2 Create a class using the Python language
2.3 Create and modify the instances and the attributes in the instance.

3. Discussion:

Object-Oriented Programming (OOP) is an approach to programming that views the world and systems as consisting of objects that relate and interact with each other. This involves identifying the characteristics that describe the object which are known as the Attributes of the object. Furthermore, it also deals with identifying the possible capabilities or actions that an object is able to do which are called Methods.

An object is simply composed of Attributes and Methods wherein Attributes are variables that hold the information describing the object and Methods are functions which allow the object to perform its defined capabilities/actions. A UML Class Diagram is used to formally represent the collection of Attributes and Methods.

An example is given below considering a simple banking system.

Accounts ATM
+ account_number: int + serial_number: int
+ account_firstname: string
+ account_lastname: string
+ current_balance: float
+ address: string + deposit(account: Accounts, amount: int) + email: string + widthdraw(account: Accounts, amount: int) + update_address(new_address: string) + check_currentbalance(account: Accounts) + update_email(new_email: string) + view_transactionsummary()

4. Materials and Equipment:

Desktop Computer with Anaconda
Python Windows Operating System

5. Procedure:

Creating Classes

1. Create a folder named **OOPIntro_LastName**
2. Create a Python file inside the **OOPIntro_LastName** folder named **Accounts.py** and copy the code shown below:

```
"""
    Accounts.py
"""

class Accounts(): # create the class
    account_number = 0
    account_firstname = ""
    account_lastname = ""
    current_balance = 0.0
    address = ""
    email = ""

    def update_address(new_address):
        Accounts.address = new_address

    def update_email(new_email):
        Accounts.email = new_email
```

3.Modify the Accounts.py and add ***self,*** before the new_address and new_email.
4.Create a new file named ATM.py and copy the code shown below:

```
"""
    ATM.py
"""

class ATM():
    serial_number = 0

    def deposit(self, account, amount):
        account.current_balance = account.current_balance + amount
        print("Deposit Complete")

    def widthdraw(self, account, amount):
        account.current_balance = account.current_balance - amount
        print("Widthdraw Complete")

    def check_currentbalance(self, account):
        print(account.current_balance)
```

Creating Instances of Classes

5.Create a new file named main.py and copy the code shown below:

```python
"""
    main.py
"""
import Accounts

Account1 = Accounts.Accounts() # create the instance/object

print("Account 1")
Account1.account_firstname = "Royce"
Account1.account_lastname = "Chua"
Account1.current_balance = 1000
Account1.address = "Silver Street Quezon City"
Account1.email = "roycechua123@gmail.com"

print(Account1.account_firstname)
print(Account1.account_lastname)
print(Account1.current_balance)
print(Account1.address)
print(Account1.email)

print()

Account2 = Accounts.Accounts()
Account2.account_firstname = "John"
Account2.account_lastname = "Doe"
Account2.current_balance = 2000
Account2.address = "Gold Street Quezon City"
Account2.email = "johndoe@yahoo.com"

print("Account 2")
print(Account2.account_firstname)
print(Account2.account_lastname)
print(Account2.current_balance)
print(Account2.address)
print(Account2.email)
```

6. Run the main.py program and observe the output. Observe the variables names account_firstname, account_lastname as well as other variables being used in Account1 and Account2.

```
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Admin>cd desktop

C:\Users\Admin\Desktop>cd OOPIntro_Planta

C:\Users\Admin\Desktop\OOPIntro_Planta>main.py
Account 1
Royce
Chua
1000
roycechua123@gmail.com

Account 2
John
Doe
2000
Gold Street Quezon City
johndoe@yahoo.com

C:\Users\Admin\Desktop\OOPIntro_Planta>
```

7. Modify the main.py program and add the code underlined in red.

```
1  """
2      main.py
3  """
4  import Accounts
5  import ATM
6
7  Account1 = Accounts.Accounts() # create the instance/object
8
9  print("Account 1")
10 Account1.account_firstname = "Royce"
11 Account1.account_lastname = "Chua"
12 Account1.current_balance = 1000
13 Account1.address = "Silver Street Quezon City"
14 Account1.email = "roycechua123@gmail.com"
15
```

8. Modify the main.py program and add the code below line 38.

```
print("Account 2")
print(Account2.account_firstname)
print(Account2.account_lastname)
print(Account2.current_balance)
print(Account2.address)
print(Account2.email)

# Creating and Using an ATM object
ATM1 = ATM.ATM()
ATM1.deposit(Account1,500)
ATM1.check_currentbalance(Account1)

ATM1.deposit(Account2,300)
ATM1.check_currentbalance(Account2)

```

9. Run the main.py program.

```
C:\Users\Admin\Desktop\OOPIntro_Planta>main.py
Account 1
Royce
Chua
1000
roycechua123@gmail.com

Account 2
John
Doe
2000
Gold Street Quezon City
johndoe@yahoo.com
Deposit Complete
1500
Deposit Complete
2300

C:\Users\Admin\Desktop\OOPIntro_Planta>
```

Create the Constructor in each Class

1. Modify the Accounts.py with the following code:
   Reminder: def _init_(): is also known as the constructor class

```python
"""
    Accounts.py
"""

class Accounts(): # create the class
    def __init__(self, account_number, account_firstname, account_lastname,
                 current_balance, address, email):
        self.account_number = account_number
        self.account_firstname = account_firstname
        self.account_lastname = account_lastname
        self.current_balance = current_balance
        self.address = address
        self.email = email

    def update_address(self,new_address):
        self.address = new_address

    def update_email(self,new_email):
        self.email = new_email
```

2. Modify the main.py and change the following codes with the red line. Do not remove the other codes in the program.

```
"""
    main.py
"""
import Accounts
import ATM

Account1 = Accounts.Accounts(account_number=123456,account_firstname="Royce",
                             account_lastname="Chua",current_balance = 1000,
                             address = "Silver Street Quezon City",
                             email = "roycechua123@gmail.com")

print("Account 1")
print(Account1.account_firstname)
print(Account1.account_lastname)
print(Account1.current_balance)
print(Account1.address)
print(Account1.email)

print()

Account2 = Accounts.Accounts(account_number=654321,account_firstname="John",
                             account_lastname="Doe",current_balance = 2000,
                             address = "Gold Street Quezon City",
                             email = "johndoe@yahoo.com")

```

3. Run the main.py program again and run the output.

```
C:\Users\Admin\Desktop\OOPIntro_Planta>main.py
Account 1
Royce
Chua
1000
roycechua123@gmail.com

Account 2
John
Doe
2000
Gold Street Quezon City
johndoe@yahoo.com
Deposit Complete
1500
Deposit Complete
2300

C:\Users\Admin\Desktop\OOPIntro_Planta>
```

6. Supplementary Activity:

Tasks

1. Modify the ATM.py program and add the constructor function.

```
# -*- coding: utf-8 -*-
"""ATM.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1c1fkckyYjKziJ7PoCJ8e-J-9xJyKunb6
"""

class ATM:
  def __init__ (self, serial_number):
    self.serial_number = serial_number

  def deposit(self, account, amount):
    account.current_balance = account.current_balance + amount
    print("Deposit Complete")

  def widthdraw(self, account, amount):
    account.current_balance = account.current_balance - amount
    print("Widthdraw Complete")

  def check_currentbalance(self, account):
    print(account.current_balance)
```

2. Modify the main.py program and initialize the ATM machine with any integer serial number combination and display the serial number at the end of the program.

```
ATM1 = ATM.ATM(serial_number = 12345)
ATM1.deposit(Account1, 500)
ATM1.check_currentbalance(Account1)
print(ATM1.serial_number)

ATM2 = ATM.ATM(serial_number = 54321)
ATM2.deposit(Account2, 300)
ATM2.check_currentbalance(Account2)
print(ATM2.serial_number)
```

```
C:\Users\Admin\Desktop\OOPIntro_Planta>main.py
Account 1
Royce
Chua
1000
roycechua123@gmail.com

Account 2
John
Doe
2000
Gold Street Quezon City
johndoe@yahoo.com
Deposit Complete
1500
12345
Deposit Complete
2300
54321
```

3. Modify the ATM.py program and add the **view_transactionsummary()** method. The method should display all the transactions made in the ATM object.

```
class ATM:
  def __init__ (self, serial_number):
    self.serial_number = serial_number
    self.transactions = []

  def deposit(self, account, amount):
    account.current_balance = account.current_balance + amount
    self.transactions.append(f"Desposited {amount} to {account.account_firstname} {account.account_lastname}'s account")
    print("Deposit Complete")

  def widthdraw(self, account, amount):
    account.current_balance = account.current_balance - amount
    self.transactions.append(f"Widthdrawn {amount} to {account.account_firstname} {account.account_lastname}'s account")
    print("Widthdraw Complete")

  def check_currentbalance(self, account):
    print(account.current_balance)

  def view_transactionsummary(self):
    for transaction in self.transactions:
      print(transaction)


ATM1.view_transactionsummary()
print()
ATM2.view_transactionsummary()
```

```
C:\Users\Admin\Desktop\OOPIntro_Planta>main.py
Account 1
Royce
Chua
1000
roycechua123@gmail.com

Account 2
John
Doe
2000
Gold Street Quezon City
johndoe@yahoo.com
Deposit Complete
1500
12345
Deposit Complete
2300
54321
Desposited 500 to Royce Chua's account

Desposited 300 to John Doe's account

C:\Users\Admin\Desktop\OOPIntro_Planta>
```

Questions

1. What is a class in Object-Oriented Programming?
   A class is a template declaration of the variables and methods in a certain type of object in object-oriented programming. Consequently, an object is a particular instance of a class that doesn't contain variables but rather real values.

2. Why do you think classes are being implemented in certain programs while some are sequential(line-by-line)?
   In this case, we used classes because this is a quite complex program involving more than just one python file. Sequential programming is typically suited for small, straightforward tasks where execution happens step-by-step without needing much organization or reuse, which is not fitting for this program.

3. How is it that there are variables of the same name such as account_firstname and account_lastname that exist but have different values?
   Because the said variables are considered instance variables. By using the self variable inside the _ init _ special class method, we are able to have a variable have different values .

4. Explain the constructor functions role in initializing the attributes of the class? When does the Constructor function execute or when is the constructor function called?
The constructor function __init__ is responsible for initializing an object's attributes when the object is created. It allows you to pass arguments when creating an object and assign those arguments to the object's instance variables.

5. Explain the benefits of using Constructors over initializing the variables one by one in the main program

By using constructors for initializing variables, we can declare our variables and initialize their attributes in an organized way, making it less confusing. With the help of the self variable inside the __init__ special class method, we can also set different values for the same variable.

7. Conclusion:

The program effectively demonstrates the concepts of OOP by using classes to create a simulation of ATMs. The use of the Accounts class allows for the creation of multiple bank accounts, each with its own attributes such as account_number, account_firstname, current_balance, and email. This emphasizes the details of each account within distinct objects.

8. Assessment Rubric: