Laboratory Activity 1 - Class, Objects, Methods	
Course Code: CPE009B	Program: BSCPE
Course Title: Object-Oriented Programming	Date Performed: 09/14/24
Name: Rodriguez, Kyle Franz P.	Date Submitted: 09/14/24
Section: CPE21S4	Instructor: Maria Rizette Sayo

Procedure

Code

```
"""Accounts"""

class Accounts():
    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


"""ATM"""

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("Withdraw Complete")

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

```python
"""Main"""
import Accounts
import ATM

#Create Account
Account1 = Accounts.Accounts()
Account1.account_firstname = "Royce"
Account1.account_lastname = "Chua"
Account1.current_balance = 1000
Account1.address = "Silver Street Quezon City"
Account1.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()
Account2.account_firstname = "John"
Account2.account_lastname = "Doe"
Account2.current_balance = 2000
Account2.address = "Gold Street Quezon City"
Account2.email = "johndoe@yahoo.com"
print("Account2")
print(Account2.account_firstname)
print(Account2.account_lastname)
print(Account2.current_balance)
print(Account2.address)
print(Account2.email)
print("")
#Create ATM
print("-ATM-")
print("Account1")
ATM1 = ATM.ATM()
ATM1.deposit(Account1, 500)
ATM1.check_balance(Account1)
print("")
ATM2 = ATM.ATM()
ATM2.deposit(Account2, 300)
ATM2.check_balance(Account2)
```

Output

```
C:\Users\Kyle\Desktop\OOPBintro_Rodriguez>python Main.py
Account 1
Royce
Chua
1000
Silver Street Quezon City
roycechua123@gmail.com

Account2
John
Doe
2000
Gold Street Quezon City
johndoe@yahoo.com

-ATM-
Account1
Deposit Complete
1500

Deposit Complete
2300

C:\Users\Kyle\Desktop\OOPBintro_Rodriguez>
```

Supplementary Activity

Tasks	Code
1	

```python
"""Accounts"""

class Accounts():

    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(new_address):
        Accounts.address = new_address

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

```python
# -*- coding: utf-8 -*-
"""ATM"""

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("Withdraw Complete")

    def check_balance(self, account):
        print("Current balance:",account.current_balance)
```

2	

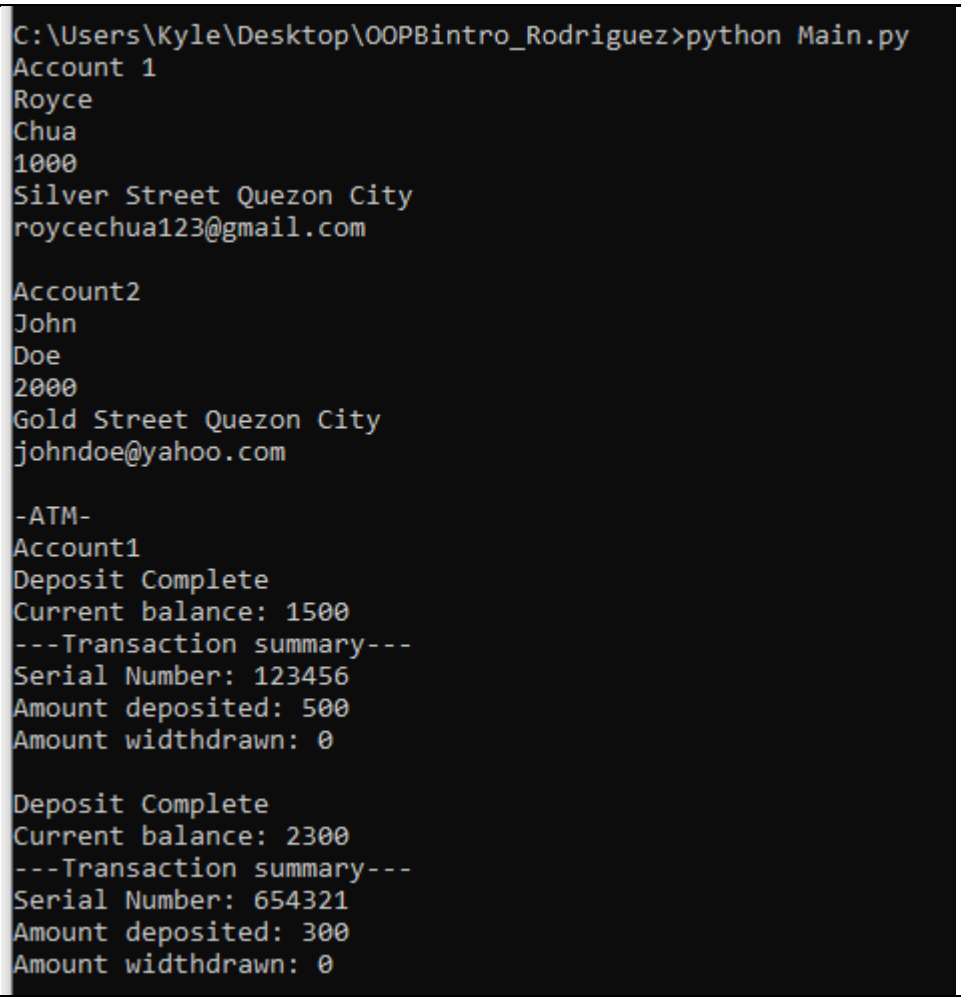
```python
# -*- coding: utf-8 -*-
"""ATM"""

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("Withdraw Complete")

    def check_balance(self, account):
        print("Current balance:",account.current_balance)

    def view_transactionsummary(self, deposit, widthdraw):
        print("---Transaction summary---")
        print("Serial Number:",self.serial_number)
```

3

```python
# -*- coding: utf-8 -*-
"""ATM"""

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("Withdraw Complete")

    def check_balance(self, account):
        print("Current balance:",account.current_balance)

    def view_transactionsummary(self, deposit, widthdraw):
        print("---Transaction summary---")
        print("Serial Number:",self.serial_number)
        print("Amount deposited:", deposit)
        print("Amount widthdrawn:",widthdraw)
```

Main Driver

```python
"""Main"""

import Accounts
import ATM

#Create Account
Account1 = Accounts.Accounts(123456,"Royce","Chua",1000,"Silver Street Quezon City","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(654321,"John","Doe",2000,"Gold Street Quezon City","johndoe@yahoo.com")


print("Account2")
print(Account2.account_firstname)
print(Account2.account_lastname)
print(Account2.current_balance)
print(Account2.address)
print(Account2.email)
print("")


#Create ATM
print("-ATM-")
print("Account1")
ATM1 = ATM.ATM(123456)
ATM1.deposit(Account1, 500)
ATM1.check_balance(Account1)
ATM1.view_transactionsummary(500, 0)
print("")
ATM2 = ATM.ATM(654321)
ATM2.deposit(Account2, 300)
ATM2.check_balance(Account2)
ATM2.view_transactionsummary(300, 0)
```

Output	
Q1	***"What is a class in Object-Oriented Programming?"*** I would define a class in Object-Oriented Programming a property of an object. It is what makes an object an object. I mean that, based on the activity that we had, it looks like an organization of data in an object. Like, say for example, the main class is named "glass". Now what makes the "glass" glass is its properties, which we organize via set of methods and variables defining it.
Q2	***"Why do you think classes are being implemented in certain programs while some are sequential(line-by-line)?"*** In OOP, I think classes are being implemented in certain programs to create an organized structure like a read object with a property and behavior that interacts with a sequential line. Speaking of sequential line, I think some codes are in this structure because it is easier to create a sequence of algorithms.
Q3	***"How is it that there are variables of the same name such "account_firstname" and "account_lastname" that exist but have different values?"*** Well, if we are talking about syntax, these two variables are different because they have different names. In the code, we initialize what value they will get based on typing their name alongside their data type and value.

Q4	***"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?"*** In OOP, the constructor function initializes the attributes of the class automatically whenever a new object is created under its name. That is basically what it does, the def __init__(self) function is like a placeholder of the objects that will be placed on the class, and with the help of the "self" keyword, you can use that for the functions under the class so that you also take the attributes for it.
Q5	***"Explain the benefits of using Constructors over initializing the variables one by one in the main program?"*** There are many benefits of using constructors. Aside from automatically initializing the attributes of a class: it makes the structure more organized; ensures that all objects of the class are consistently initialized; it helps with the ease of change when you need to change the logic for initialization; and it can make your code easier to read.

Conclusion

In this activity, I learned how to make and apply classes in python language. In the procedure, the classes named Account and ATM showed how we can make real object with their own properties and behavior with the use of the different attributes. However, the codes were not as clean since we did not utilize the function of a constructor. In the supplementary activity, we were tasked to modify the code and use a constructor. In conclusion, I learned how classes work.