Laboratory Activity No. 1

Introduction to Object-Oriented Programming

Course Code: CPE009B	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Perform:** **Aug 29, 2024**
Section: CPE21S1	**Date Submitted: August 29, 2024**
Name: Katarina Nicole R. Delavin	**Instructor: Engr. Maria Rizette 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()

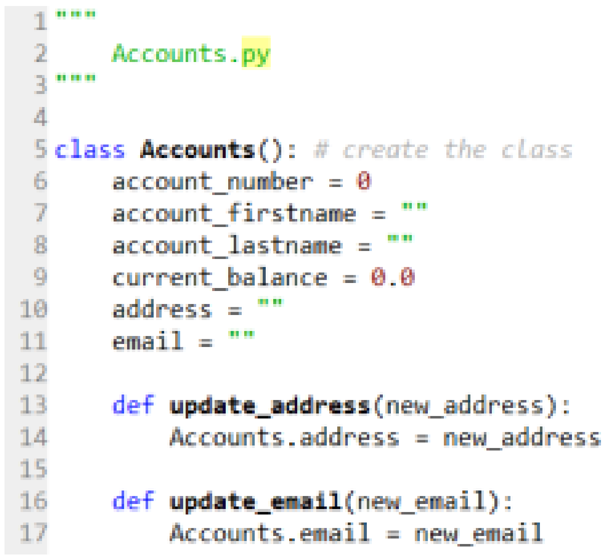
4. Materials and Equipment:

Desktop Computer with Anaconda Python
Windows Operating System

5. Procedure:

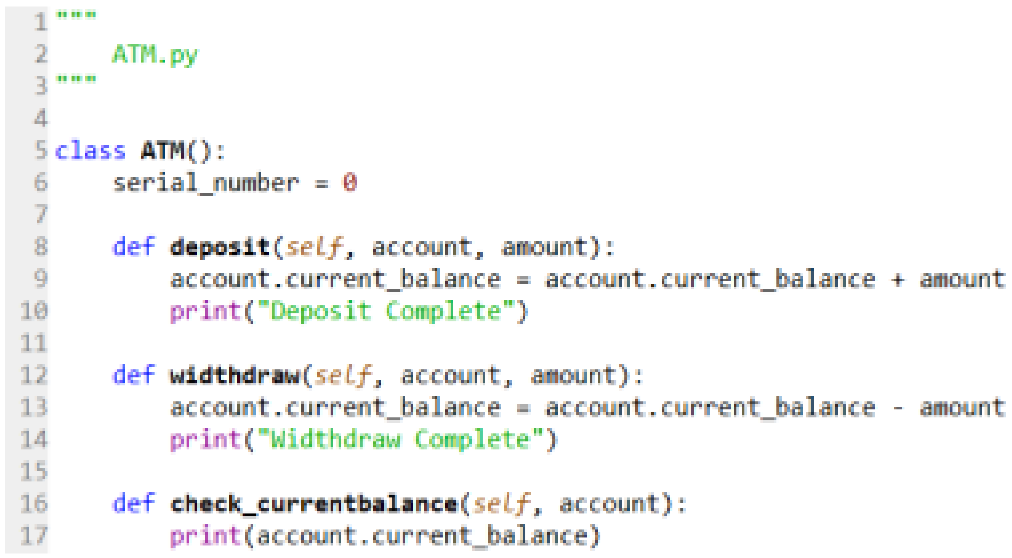
Creating Classes

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:



```python
"""
    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
```

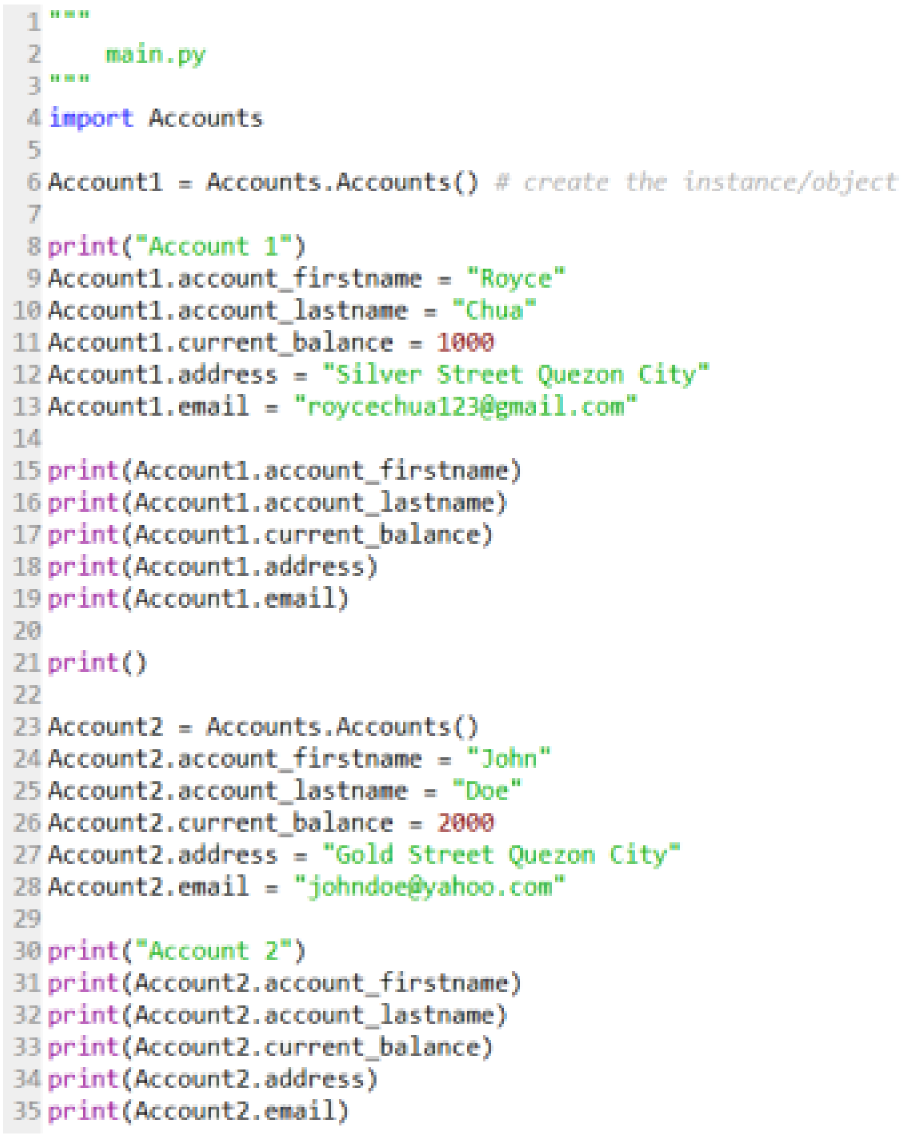
3. Modify the Accounts.py and add ***self,*** before the new_address and new_email.
4. Create a new file named ATM.py and copy the code shown below:



```python
"""
    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)
```

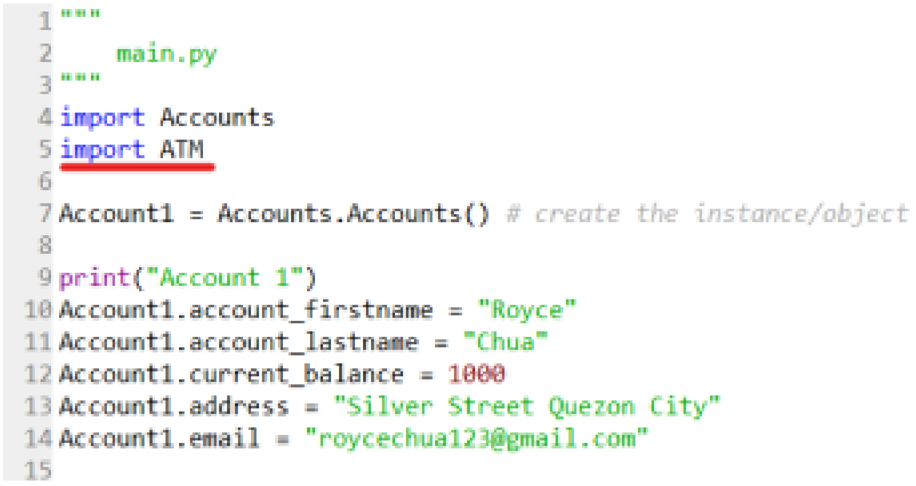
Creating Instances of Classes

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



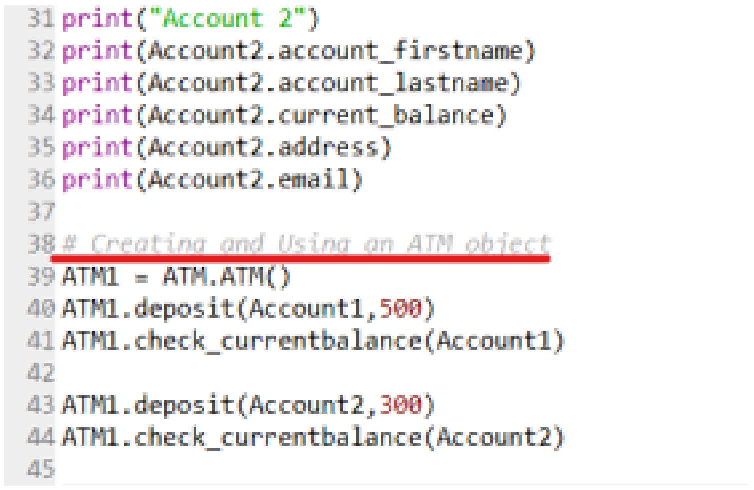
```python
"""
    main.py
"""
import Accounts

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 the Account1 and Account2. 7. Modify the main.py program and add the code underlined in red.



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

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"

```

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



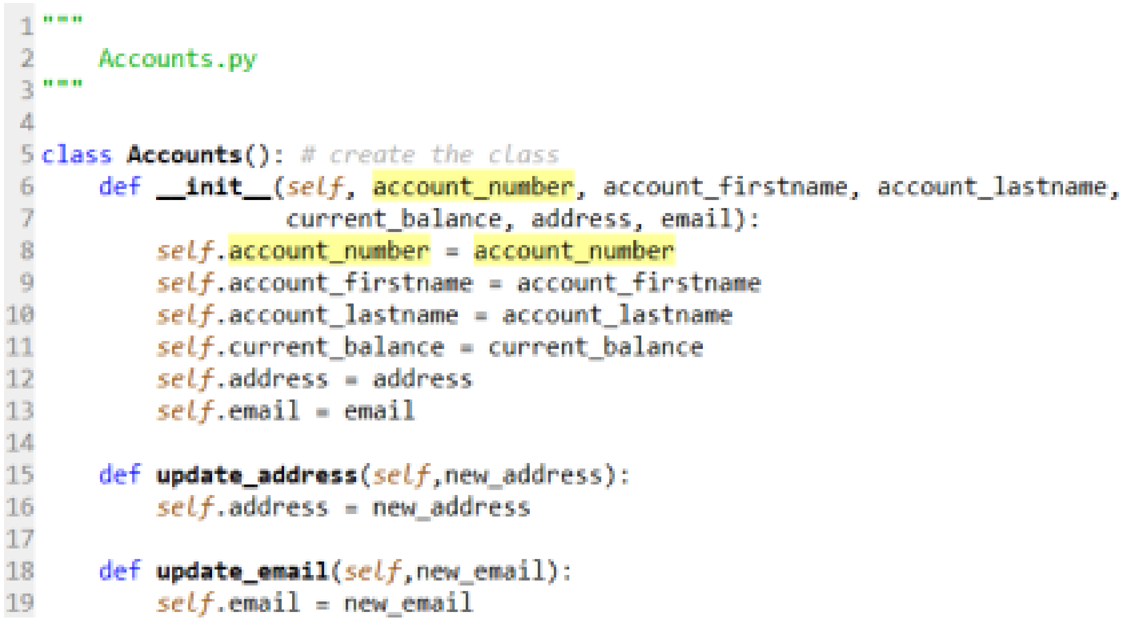
```
print("Account 2")
print(Account2.account_firstname)
print(Account2.account_lastname)
print(Account2.current_balance)
print(Account2.address)
print(Account2.email)

# 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.

Create the Constructor in each Class

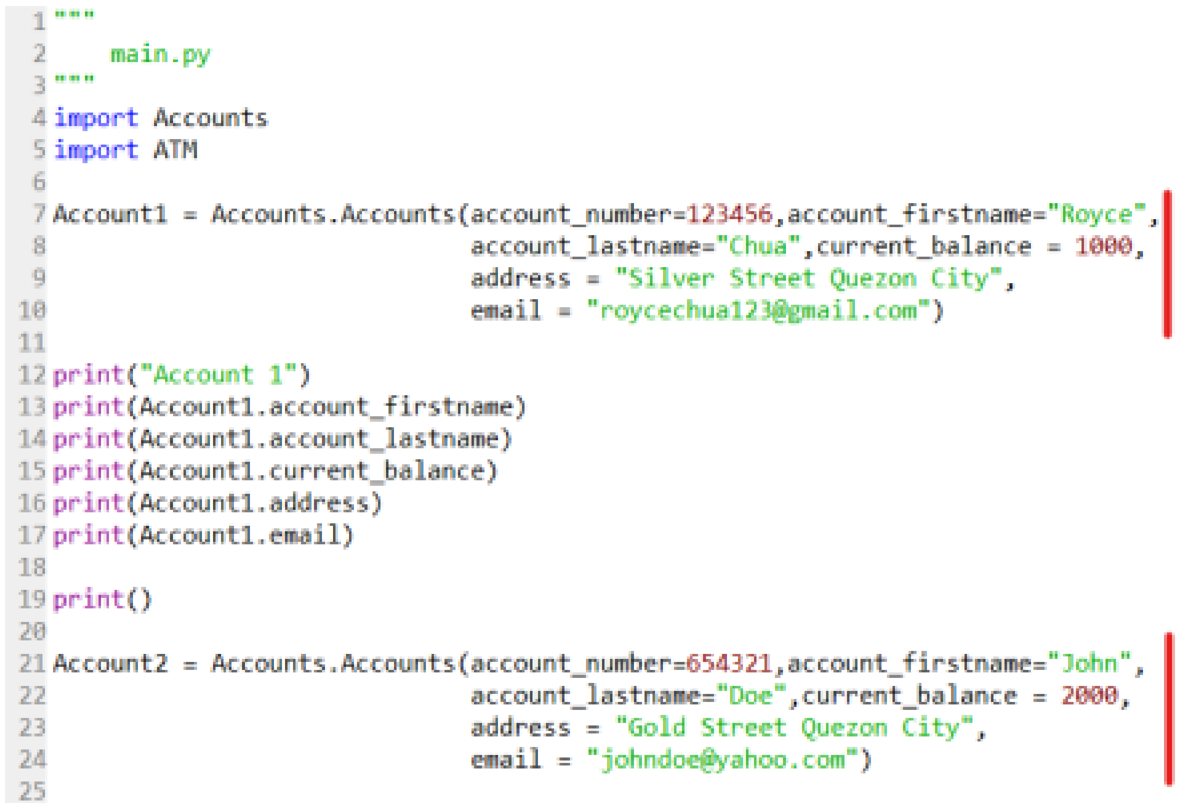
1. Modify the Accounts.py with the following code:

   Reminder: def __init__(): is also known as the constructor class



```
"""
    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.



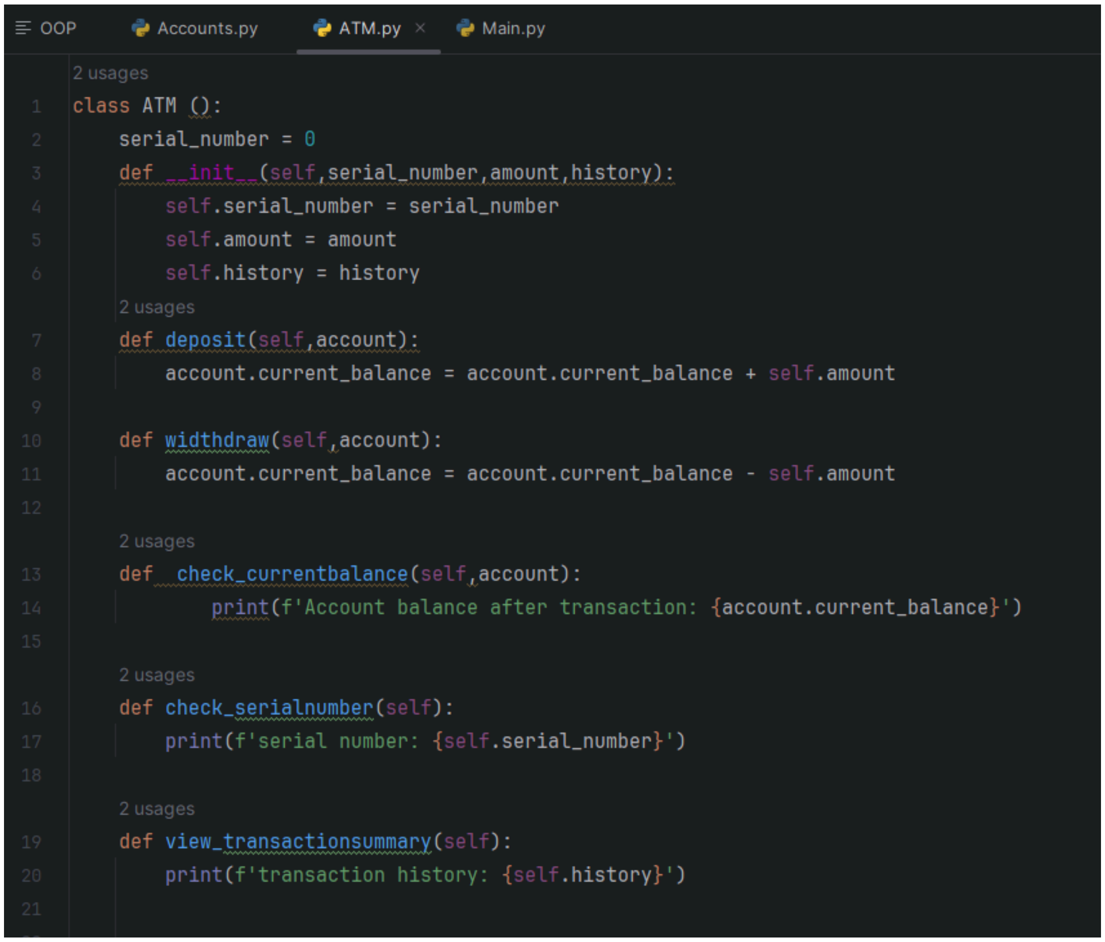
```python
"""
    main.py
"""
import Accounts
import ATM

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.

6. Supplementary Activity:

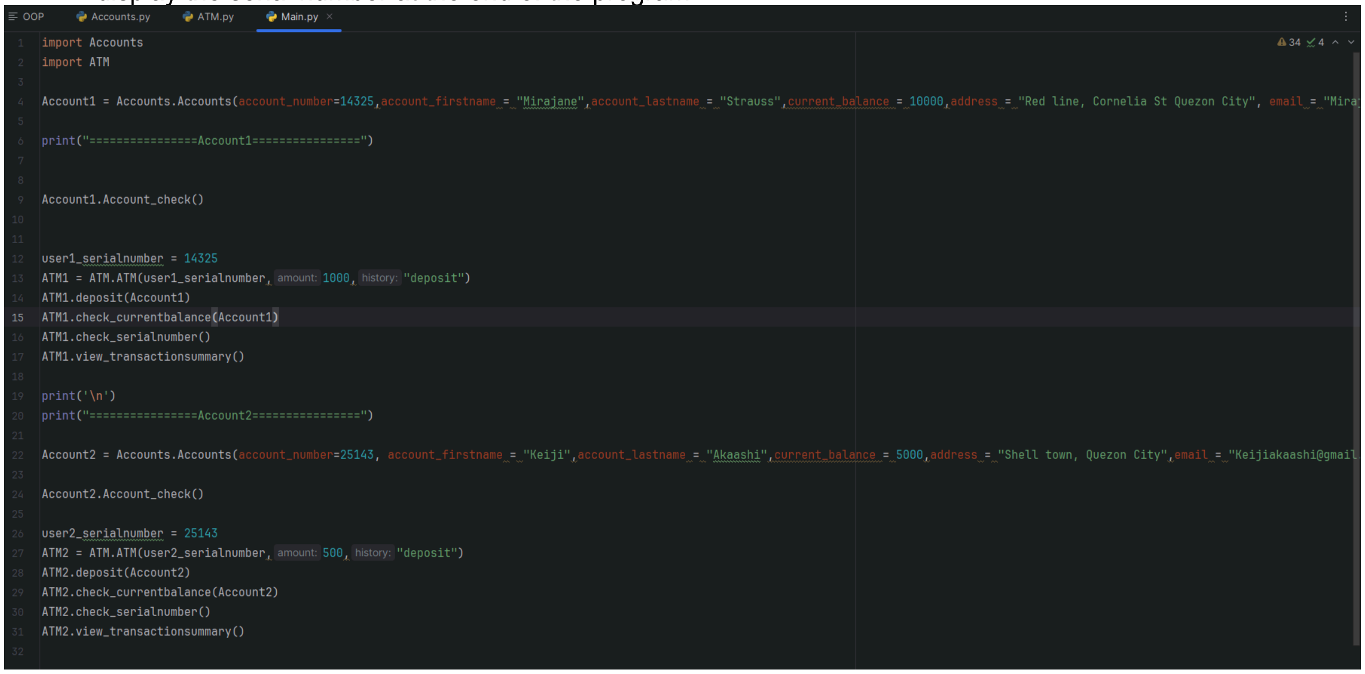
Tasks

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

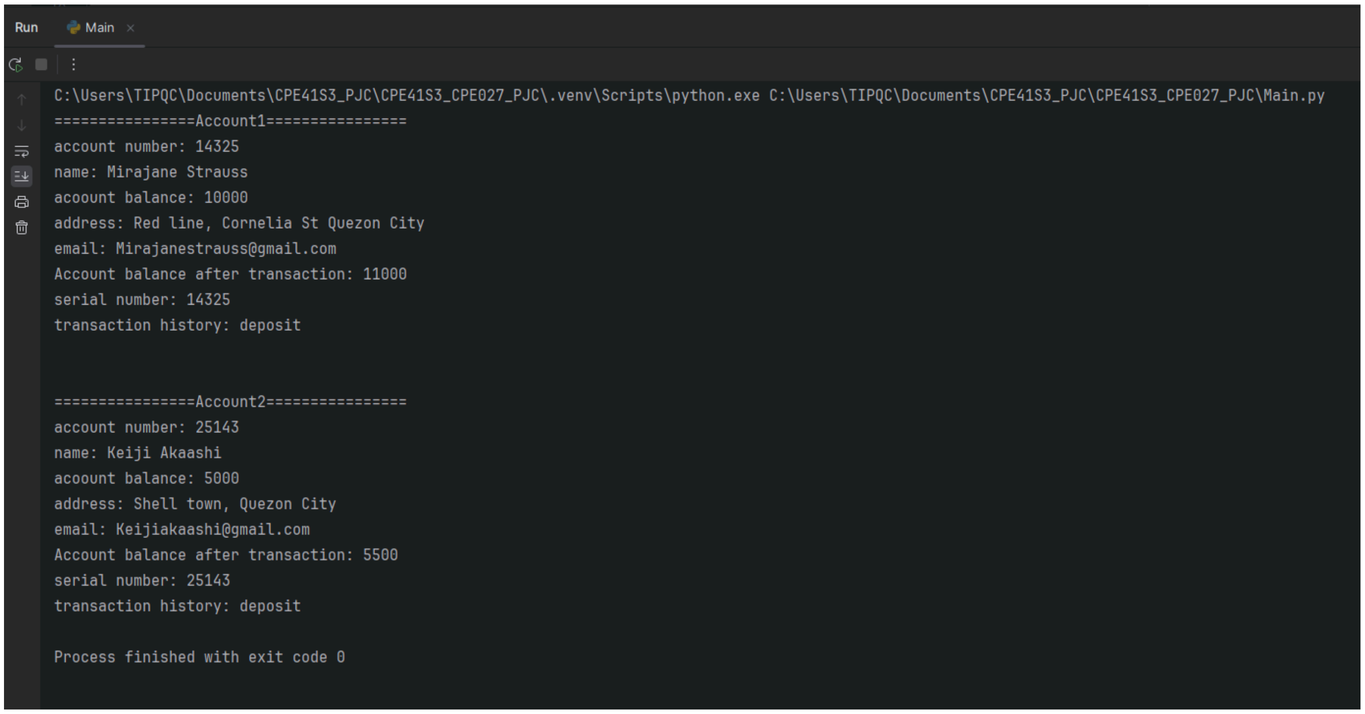


```python
2 usages
class ATM ():
    serial_number = 0
    def __init__(self,serial_number,amount,history):
        self.serial_number = serial_number
        self.amount = amount
        self.history = history
    2 usages
    def deposit(self,account):
        account.current_balance = account.current_balance + self.amount

    def widthdraw(self,account):
        account.current_balance = account.current_balance - self.amount

    2 usages
    def  check_currentbalance(self,account):
            print(f'Account balance after transaction: {account.current_balance}')

    2 usages
    def check_serialnumber(self):
        print(f'serial number: {self.serial_number}')

    2 usages
    def view_transactionsummary(self):
        print(f'transaction history: {self.history}')
```

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.



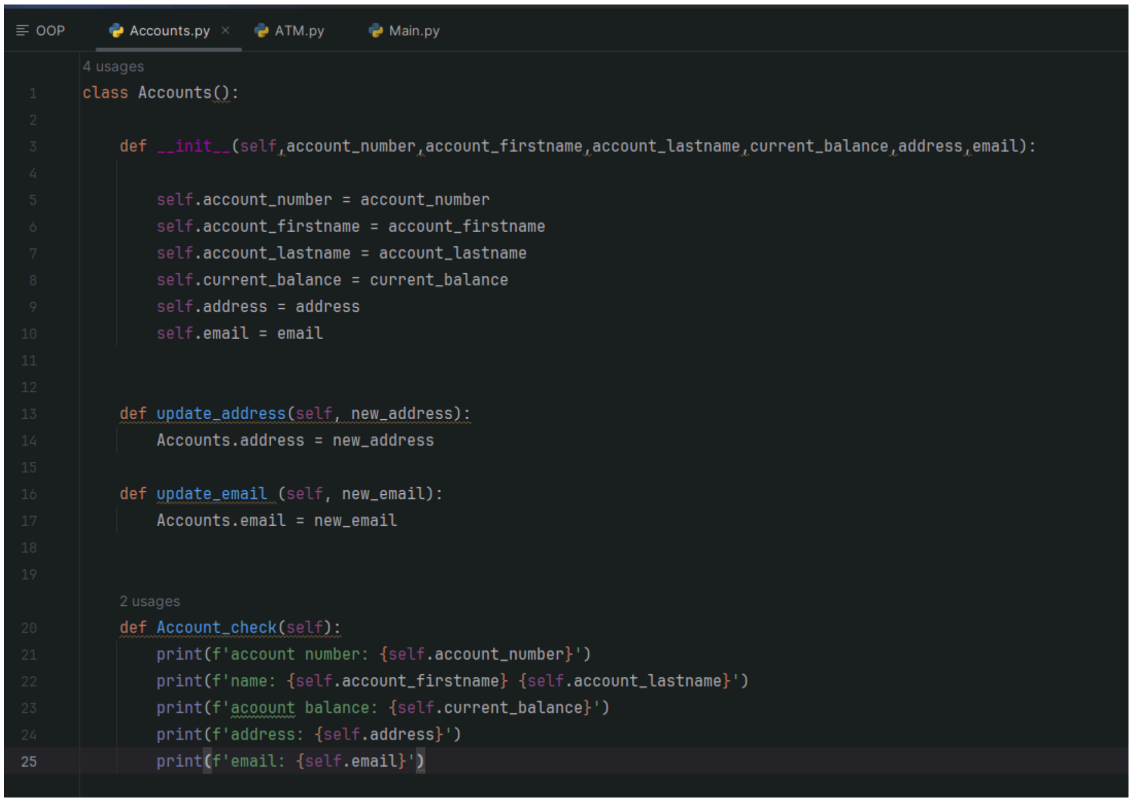
```python
import Accounts
import ATM

Account1 = Accounts.Accounts(account_number=14325,account_firstname = "Mirajane",account_lastname = "Strauss",current_balance = 10000,address = "Red line, Cornelia St Quezon City", email = "Mira

print("===============Account1===============")


Account1.Account_check()


user1_serialnumber = 14325
ATM1 = ATM.ATM(user1_serialnumber, amount: 1000, history: "deposit")
ATM1.deposit(Account1)
ATM1.check_currentbalance(Account1)
ATM1.check_serialnumber()
ATM1.view_transactionsummary()

print('\n')
print("===============Account2===============")

Account2 = Accounts.Accounts(account_number=25143, account_firstname = "Keiji",account_lastname = "Akaashi",current_balance = 5000,address = "Shell town, Quezon City",email = "Keijiakaashi@gmail

Account2.Account_check()

user2_serialnumber = 25143
ATM2 = ATM.ATM(user2_serialnumber, amount: 500, history: "deposit")
ATM2.deposit(Account2)
ATM2.check_currentbalance(Account2)
ATM2.check_serialnumber()
ATM2.view_transactionsummary()
```



Run Main

```
C:\Users\TIPQC\Documents\CPE41S3_PJC\CPE41S3_CPE027_PJC\.venv\Scripts\python.exe C:\Users\TIPQC\Documents\CPE41S3_PJC\CPE41S3_CPE027_PJC\Main.py
================Account1================
account number: 14325
name: Mirajane Strauss
acoount balance: 10000
address: Red line, Cornelia St Quezon City
email: Mirajanestrauss@gmail.com
Account balance after transaction: 11000
serial number: 14325
transaction history: deposit


================Account2================
account number: 25143
name: Keiji Akaashi
acoount balance: 5000
address: Shell town, Quezon City
email: Keijiakaashi@gmail.com
Account balance after transaction: 5500
serial number: 25143
transaction history: deposit

Process finished with exit code 0
```

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



OOP Accounts.py ATM.py Main.py

```
4 usages
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(self, new_address):
        Accounts.address = new_address

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


    2 usages
    def Account_check(self):
        print(f'account number: {self.account_number}')
        print(f'name: {self.account_firstname} {self.account_lastname}')
        print(f'acoount balance: {self.current_balance}')
        print(f'address: {self.address}')
        print(f'email: {self.email}')
```

Questions

1. What is a class in Object-Oriented Programming?
    - In object-oriented programming, a class functions as a blueprint that specifies the structures and behavior, such as methods, and may be used as a template for creating cases, allowing you to model actual things in code.

2. Why do you think classes are being implemented in certain programs while some are sequential(line-by-line)?
    - Programs can be organized using classes to make code reusable, while sequential (line-by-line) programs follow a more easy execution flow that does not require reusable or object structure.

3. How is it that there are variables of the same name such account_firstname and account_lastname that exist but have different values?
    - Even though they are similar in naming convention, variables like "account_firstname" and "account_lastname" indicate separate bit values of data connected to various properties, therefore they might 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, which is run automatically at object instantiation, sets the initial values for a class's attributes at object creation.

5. Explain the benefits of using Constructors over initializing the variables one by one in the main program?
    - Compared to manually initializing each variable in the main program, using constructors enables effective, consistent, and organized initialization of variables, avoiding or minimizing repetition and potential errors.

7. Conclusion:

In summary, the class function in object-oriented programming increases the specificity of our program. Constructors and classes set up objects with defined values and functions, making it possible to write structured, reusable, and modular code.
.

8. Assessment Rubric: