Laboratory Activity No. 1	
Introduction to Object-Oriented Programming	
Course Code: CPE009B	**Program: BSCPE**
Course Title: Object-Oriented Programming	**Date Performed:** 9/14/24
Section: CPE21S4	**Date Submitted:** 9/15/24
Name: Santos, Andrei R.	**Instructor:** Professor Maria Rizette Sayo

6. Supplementary Activity:

Tasks:

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

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

class ATM:

   def __init__(self, serial_number = 0, personal_account = None):
      self.serial_number = serial_number
      self.personal_account = personal_account
      self.transactions = []

   def deposit(self, amount):
      self.personal_account.current_balance += amount
      self.transactions.append(f"The amount that was deposited is {amount}")
      print("\nDeposit Complete")

   def withdraw(self, amount):
      if amount > self.personal_account.current_balance:
         print("Invalid transaction, not enouugh balance.")
      else:
         self.account.current_balance -= amount
         self.transactions.append(f"You withdraw {amount}")
         print("Withdrawal Complete.")

   def check_currentbalance(self):
      print(f"Your Current Balance: {self.personal_account.current_balance}")

   def view_transactionsummary(self):
      print("Transaction Summary:")
      for transaction in self.transactions:
         print(transaction)
```

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.

```
"""
    main.py
"""

import Accounts
import ATM
import random

Account1 = Accounts.Accounts(account_number = 123456, account_firstname = "Royce",
account_lastname = "Chua", current_balance = 1000, address = "Silver Sreet 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@gmail.com")
print("Account 2")
print(Account2.account_firstname)
print(Account2.account_lastname)
print(Account2.current_balance)
print(Account2.address)
print(Account2.email)
print()

atm1_serialnum = random.randint(123456, 987654)
atm_1 = ATM.ATM(serial_number=atm1_serialnum, personal_account=Account1)

atm2_serialnum = random.randint(123456, 987654)
atm_2 = ATM.ATM(serial_number=atm2_serialnum, personal_account=Account2)

atm_1.deposit(500)
print(f"ATM 1 Serial No. {atm1_serialnum}")
atm_1.check_currentbalance()

atm_2.deposit(300)
print(f"ATM 2 Serial No. {atm2_serialnum}")
atm_2.check_currentbalance()
```

```
print("\nATM 1 Transaction History:")
atm_1.view_transactionsummary()
print("\nATM 2 Transaction History:")
atm_2.view_transactionsummary()
```

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

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

class ATM:
   def __init__(self, serial_number = 0, personal_account = None):
      self.serial_number = serial_number
      self.personal_account = personal_account
      self.transactions = []

   def deposit(self, amount):
      self.personal_account.current_balance += amount
      self.transactions.append(f"The amount that was deposited is {amount}")
      print("\nDeposit Complete")

   def withdraw(self, amount):
      if amount > self.personal_account.current_balance:
         print("Invalid transaction, not enouugh balance.")
      else:
         self.account.current_balance -= amount
         self.transactions.append(f"You withdraw {amount}")
         print("Withdrawal Complete.")

   def check_currentbalance(self):
      print(f"Your Current Balance: {self.personal_account.current_balance}")

   def view_transactionsummary(self):
      print("Transaction Summary:")
      for transaction in self.transactions:
         print(transaction)
```

Questions

1. What is a class in Object-Oriented Programming?

A class helps in creating and declaring objects by providing a clear code. It defines the attributes and methods that objects will have, allowing for the use of functions like if statements or loops within its structure. Classes are particularly beneficial because they help create neat and organized code, making it easier to manage and maintain the program. By its related data and functions, classes improve code

readability and efficiency.

2. Why do you think classes are being implemented in certain programs while some are sequential(line-by-line)?

Classes also provide a straightforward approach, as some programs require simple and efficient code for basic functionality. This allows programmers to create accurate and less complex programs quickly. By using classes, users can focus on writing the needed code. This makes it easier to develop simple applications while still benefiting from using classes.

3. How is it that there are variables of the same name such account_firstname and account_lastname that exist but have different values?

Variables are case-sensitive, meaning that even small changes in their names can result in new and different values. Each variable is assigned to a specific scope or context, which helps keep track of its value. Allowing similar names for related variables helps me and other programmers connect and understand the code without conflicts. This organizing of codes ensures clarity and prevents confusion, making it easier to manage and reference the variables needed.

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 is essential for a class when an object is created, as it sets up default or user-provided values. It runs automatically during the creation of the object, ensuring that the object starts with the necessary information. This helps initialize the object's attributes properly, allowing the program to function correctly from the beginning. By using the constructor, developers can ensure that each object has the required data to operate as intended.

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

Constructors make programming quicker and easier by ensuring that important information is set up correctly, which helps avoid interpretation errors and other issues. They contribute to cleaner code by initialization within the class. When using constructors, programmers can assign values efficiently and check those values accurately, leading to more reliable and maintainable code. Overall, constructors enhance the organization and clarity of the program.

7. Conclusion:

In conclusion, understanding the principles of classes, constructors, and variables significantly enhances the ability to write organized and efficient code. By applying these concepts, I have learned how to create a simple bank transaction, ensuring that the descriptions and details provide accurate data. This knowledge allows me to produce clear and reliable final outputs in my programs. Overall, these

programming fundamentals contribute to my development as a competent programmer, enabling me to tackle more complex projects in the future.
8. Assessment Rubric: