Laboratory Activity No. 1

Introduction to Object-Oriented Programming

Course Code: CPE009B **Program:** BSCPE

Course Title: Object-Oriented Programming **Date Performed:**

Section: CpE21S4 **Date Submitted:**

Name: Esteban, Prince Wally G. **Instructor:** Mrs. Ma. 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:

```
"""
    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:

```
"""
    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.

```
"""
    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.

```
"""
    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
#atm.py
class ATM:
    def __init__(self, serial_number):
        self.serial_number = serial_number
        self.balance = 0
        self.transactions = []

    def deposit(self, amount):
        self.balance += amount
        self.transactions.append(f"Deposited {amount}")

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            self.transactions.append(f"Withdrew {amount}")
        else:
            print("Insufficient funds")

    def view_balance(self):
        print(f"Current balance: {self.balance}")

    def view_transaction_summary(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.

```python
#main.py
#from ATM import ATM

def main():
    print("Starting main function")
    atm = ATM(serial_number=123456)
    print("ATM created")

    atm.deposit(1000)
    atm.withdraw(500)
    atm.deposit(200)

    atm.view_balance()
    atm.view_transaction_summary()

    print(f"ATM Serial Number: {atm.serial_number}")


if __name__ == "__main__":
    main()
```

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

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

```
Starting main function
ATM created
Current balance: 700
Transaction Summary:
Deposited 1000
Withdrew 500
Deposited 200
ATM Serial Number: 123456
```

Questions

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

   A class in Object-Oriented Programming is like a template for creating objects. Imagine you're designing a blueprint for building cars. This blueprint includes details about what features each car will have, like its color and model, and what it can do, like driving or honking.
   When you actually build a car based on this blueprint, you're creating an object from the class.
   So, a class helps you define what an object should look like and what it can do.

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

   Classes are used in some programs because they help keep code organized and reusable. Imagine you're building a big project like a video game. Classes let you bundle together things like the properties and actions of game characters, which makes it easier to manage and update. In contrast, if you're working on something simple, like a small script, writing code line-by-line might be quicker and easier because you don't need all that extra structure. So, classes are great for complex stuff, while line-by-line code works well for simpler tasks.

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 like account_firstname and account_lastname can have different values because they might be in different places in your code. For example, if one is inside a function and another is outside, they're treated separately. Or, if they're part of different objects or classes, each one can have its own value. So, the same name can be used in different parts of your program without mixing up the 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 is like a special setup tool that runs whenever you create a new object from a class. Its job is to set up the object's initial values. For example, if you make a `Person` object, the constructor will make sure that things like `name` and `age` are set up the way you want right from the start. It's called automatically when you create the object, so you don't have to run it manually.

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

   Using constructors is better than setting up variables one by one because they automatically set up everything when you create an object. This way, you don't have to remember to initialize each variable separately, and your main code stays tidy. Constructors make sure every object starts the same way, reducing mistakes and making your code easier to read and manage. Plus, they can provide default values if you don't give any, saving you time and effort.

7. Conclusion:

In conclusion, constructors play a crucial role in initializing objects by setting up their attributes automatically when they are created. This ensures that every object starts with a consistent and well-defined state, making the code cleaner and less error-prone. While constructors offer significant benefits such as automatic and consistent initialization, cleaner code organization, and the ability to provide default values, sequential code can be simpler for straightforward tasks. By using constructors, you avoid the hassle of manually initializing each variable, which helps keep your main program organized and reduces the risk of mistakes. Overall, constructors help manage complexity and maintain code efficiency, especially in larger or more complex applications.

8. Assessment Rubric: