UNIVERSITY OF CALOOCAN CITY
Caloocan, 1400 Metro Manila, Philippines

COLLEGE OF ENGINEERING
Computer Engineering
2nd Semester, School Year 2024-2025

Laboratory Activity No. 3.1	
Introduction to Object-Oriented Programming	
Course Code: CPE103	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Performed:** 01/25/2025
Section: CPE 1-A NORTH	**Date Submitted:**01/25/2025
Name: PALMES, LEWIS CLARK L.	**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:	

UNIVERSITY OF CALOOCAN CITY
Caloocan, 1400 Metro Manila, Philippines

COLLEGE OF ENGINEERING
Computer Engineering
2nd Semester, School Year 2024-2025

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/Python Colab 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:

```
"""
    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

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

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

print("Account 1")
Account1.account_firstname = "Royce"
Account1.account_lastname = "Chua"
```

red.

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:

UNIVERSITY OF CALOOCAN CITY

Caloocan, 1400 Metro Manila, Philippines

COLLEGE OF ENGINEERING

Computer Engineering

2nd Semester, School Year 2024-2025

Tasks

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

For checking refer to this link:
PYTHON PROGRAM

Questions

1. What is a class in Object-Oriented Programming?
   A Object-Oriented Programming (OOP) class serves as a blueprint for creating objects. It defines a set of attributes and methods the objects created from the class will have. This encapsulation of data and behavior allows for modular and reusable code, promoting organized and efficient programming practices.

2. Why do you think classes are being implemented in certain programs while some are sequential(line-by-line)?
   Classes are implemented in programs to provide structure, modularity, and reusability, especially in complex applications. They encapsulate related data and functions, making the code more organized and maintainable. Sequential (line-by-line) programming is often sufficient for simpler tasks, but classes offer better state management and reduce redundancy in larger, more complex programs.

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?

   In OOP, multiple class instances can have their own set of attributes, even if they share the same attribute names. Each instance is independent, so account1.firstname and account2.firstname can hold different values. This encapsulation ensures that changes to one instance do not affect others, maintaining data integrity.

UNIVERSITY OF CALOOCAN CITY
Caloocan, 1400 Metro Manila, Philippines

COLLEGE OF ENGINEERING
Computer Engineering
2nd Semester, School Year 2024-2025

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, usually named __init__ in Python, initializes the attributes of a class when an instance is created. It sets up the object's initial state with provided or default values. The constructor is automatically executed when a new object is instantiated, ensuring proper initialization before the object is used.

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

Constructors provide consistent and complete initialization of objects, encapsulating the setup logic within the class. This results in cleaner and more maintainable code, as the main program can focus on higher-level logic. Constructors also reduce redundancy by centralizing the initialization process, improving code reusability, and reducing potential errors.

7. Conclusion: In conclusion, this laboratory activity has provided a comprehensive introduction to Object-Oriented Programming (OOP) concepts, particularly the creation and utilization of classes in Python. Through the hands-on tasks of defining classes, creating instances, and modifying attributes, we have gained a practical understanding of how OOP principles can be applied to develop modular and maintainable code. This structured programming approach allows for better organization of data and functionality, making it easier to manage and scale applications as they grow in complexity.

Furthermore, the supplementary activities have reinforced the benefits of using constructors and encapsulation in class definitions. By automating the initialization of objects and centralizing the setup logic within the class, we have observed how constructors enhance code readability, reusability, and maintainability. Overall, this activity has equipped us with the foundational skills needed to effectively utilize OOP techniques in Python, paving the way for more advanced programming challenges and real-world application development.

8. Assessment Rubric: