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:** January 25, 2025
Section: 1- A	**Date Submitted:** January 31, 2025
Name: Directo, Hannah Thea B.	**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,

```
"""
    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"

```

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.

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.
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.

Laboratory Activity No. 3.1 - Colab

Questions

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

UNIVERSITY OF CALOOCAN CITY
Caloocan, 1400 Metro Manila, Philippines

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

A class is a blueprint for creating objects in Object-Oriented Programming (OOP), a fundamental concept that enables organized and efficient program development. It includes data properties and methods serving as a template for defining object structure and behavior. Classes encourage modularity, reusability, and code organization, which makes complicated systems easier to maintain. Classes in OOP, in essence, provide a strong tool for designing and creating well-structured, manageable, and extensible software systems.

2. **Why do you think classes are being implemented in certain programs while some are sequential(line-by-line)?**
Because classes in Object-Oriented Programming help structure code, making it easier to maintain, reuse, and expand. They combine related data and functions, allow code reuse through objects, and support inheritance and abstraction. While, Sequential code, is simpler and works well for smaller tasks that don't require much organization. It's faster to write but can become difficult to manage in larger programs.

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 such as `account_firstname` and `account_lastname` can exist with the same name but hold different values because they represent different pieces of data related to the same context. For example, `account_firstname` might store the user's first name, such as "John," while `account_lastname` stores their last name, such as "Doe." Though both variables share the same "account" prefix, they represent different aspects of the user's account, so their values can differ.

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?**
A constructor function is responsible for initializing the attributes of a class when an object is created. It sets up the object's initial state by assigning values to its attributes, either through default values or values provided during object creation. The constructor is automatically called as soon as an object is created, ensuring that the object starts in a valid state. In languages like Python, this function is called `__init__`**.**

5**. Explain the benefits of using Constructors over initializing the variables one by one in the main program?**
Using constructors to initialize variables in OOP offers several advantages. They promote encapsulation by keeping initialization inside the class, making code more maintainable. Constructors ensure consistency and validation by checking input data, preventing errors. They reduce code duplication by centralizing initialization

logic, improving code clarity. Also, constructors enhance readability, simplify object creation, and provide flexibility through overloading.

7. Conclusion:

In conclusion, Object-Oriented Programming (OOP) organizes complex systems into objects that interact through attributes and methods, making code more modular, maintainable, and reusable. Classes act as blueprints for creating objects, as shown in the banking system example where the `Accounts` class defines attributes like `account_number` and `current_balance` alongside methods such as `deposit` and `withdraw`. While sequential code may be simpler, classes provide a more structured solution for managing larger projects. Attributes like `account_firstname` and `account_lastname` can hold different values despite sharing the same context, highlighting the flexibility of classes. The constructor initializes these attributes when an object is created, ensuring valid starting values and reducing errors. By using constructors instead of manually initializing variables, developers can promote encapsulation, consistency, and code clarity, as demonstrated in the banking system, where the constructor simplifies the creation of account objects. Ultimately, OOP, through the use of classes, attributes, methods, and constructors, enables more organized, scalable, and efficient programming.

8. Assessment Rubric: