Laboratory Activity 1 - Class, Objects, Methods	
Name: Reyes, Alexzander J.	**09/15/2024**
Course/Section:	**Dr. Maria Rizette Sayo**

6. Supplementary Activity

TASKS

Modify the ATM.py program and add the constructor function

main.py Share Run

```
# (1) ATM.py - Add the constructor function

class ATM:
    def __init__(self, serial_number):
        # Initializing the ATM object with a serial number and an emp
            transaction history
        self.serial_number = serial_number
        self.transaction_history = []  # To store all transactions

    def deposit(self, amount):
        # Deposit money to the ATM and record the transaction
        if amount > 0:
            self.transaction_history.append(f"Deposited: ${amount}")
            print(f"${amount} has been deposited.")
        else:
            print("Invalid deposit amount.")

    def withdraw(self, amount):
        # Withdraw money from the ATM and record the transaction
        if amount > 0:
            self.transaction_history.append(f"Withdrew: ${amount}")
            print(f"${amount} has been withdrawn.")
        else:
            print("Invalid withdrawal amount.")

```

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
# (2) main.py - Initialize the ATM with a serial number and display i

def main():
    # Initializing the ATM machine with a serial number
    atm = ATM(serial_number=983486132)  # Example serial number

    # Example transactions
    atm.deposit(1000)  # Depositing $100
    atm.withdraw(423)  # Withdrawing $50

    # (3) Viewing the transaction summary
    atm.view_transactionsummary()

    # (2) Display the ATM serial number
    print(f"\nATM Serial Number: {atm.serial_number}")

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

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

```python
    # (3) Add the view_transactionsummary() method
    def view_transactionsummary(self):
        # Display the transaction summary
        if not self.transaction_history:
            print("No transactions have been made.")
        else:
            print("Transaction Summary:")
            for transaction in self.transaction_history:
                print(transaction)
```

QUESTIONS:

1. What is Objected-Oriented Programming?

The paradigm of object-oriented programming (OOP) is founded on the idea of "objects," which are entities that may hold both code and data. Code is contained in methods (or functions) that manipulate the data, whereas data is contained in attributes (or properties) within objects. By modeling real-world items and connections, object-oriented programming (OOP) makes programs more manageable, reusable, and modular. The four main tenets of OOP are abstraction, polymorphism, inheritance, and encapsulation. OOP seeks to enhance the organization, scalability, and maintenance of code by grouping it into objects that correspond to actual entities.

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

Programs construct classes to benefit from OOP concepts like polymorphism, inheritance, and encapsulation. They are especially helpful in complicated projects where functionality and data must be arranged into manageable, reusable chunks. Classes facilitate the process of constructing object blueprints, which helps with handling complicated data relationships, modeling real-world scenarios, and code maintenance. Conversely, smaller or simpler applications with less complicated data management and functionality typically employ sequential (line-by-line) programming. When the sophisticated features offered by OOP are not required for linear tasks, sequential programming is a simple and appropriate solution.

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

Same-named variables can exist in many contexts or scopes in object-oriented programming. As an illustration, account_firstname and account_lastname could belong to separate classes or objects. Although two variables may have the same name, they may belong to separate objects or classes and have different values since each object or class keeps its own instance of variables. Since variables with the same name can be accessible through their respective objects or classes, this encapsulation makes it possible for them to live without interfering with one another.

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?

When a new instance of the class is created, a particular method known as the constructor function is automatically invoked. Its main responsibility is to initialize the class's attributes, assigning default or given values to them. The constructor function establishes any required dependencies or resources and makes sure the object is initialized in a correct state. In languages like Python, Java, or C++, it is called immediately upon the creation of an object using the new

keyword. This automatic invocation aids in initializing the object with the necessary settings and getting it ready for usage.

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

When initializing variables, using constructors has a number of important advantages to establishing them manually in the main program. Constructors encapsulate setup logic within the class itself, ensuring that an object's attributes are populated consistently and logically. This method streamlines object creation, lowers errors by preventing uninitialized variables, and encourages code reuse by grouping initialization operations together. Constructors improve maintainability by centralizing initialization code, which makes it simpler to manage and update object characteristics without interfering with the main program.

CONCLUSION

In conclusion, the code manages ATM transactions using a class-based structure, illustrating the application of Object-Oriented Programming (OOP) principles. The ATM-related properties and methods are encapsulated in the class, which offers a logical and structured approach to managing deposits, withdrawals, and transaction summaries. The ATM's transaction history and serial number are initialized by constructors, guaranteeing that every instance begins in a legitimate state. It is easier to manage complicated functionalities inside the program by centralizing initialization in the constructor, which also makes the code more modular, minimizes redundancy, and improves maintainability.