OBJECT-ORIENTED PROGRAMMING (OOP) TASK IN PYTHON - PART 2

Objective:

In Part 2 of this task, you will build upon the Object-Oriented Python program developed in Part 1 to create a simple banking system that manages multiple customers and their accounts.

Instructions:

Step 1:

Create a Python class named `Bank` that represents the banking system. The `Bank` class should have the following attributes and methods:

Attributes:

- `name` (str): The name of the bank.
- `customers` (list): A list to store customer objects.

Methods:

- `__init__(self, name)`: The constructor method that initializes the `name` of the bank and an empty `customers` list.
- `add_customer(self, customer)`: A method that adds a customer object to the `customers` list.
- `get_customer(self, account_number)`: A method that returns a customer object based on the provided `account_number`. If the account number is not found, return `None`.
- `__str__(self)`: A method that returns a string representation of the bank's information, including its name and the list of customers.

Step 2:

Enhance the program by creating an instance of the `Bank` class and adding customer objects to it. Perform transactions using the `Transaction` class as before, but this time, associate transactions with the corresponding customers. Here's an example:

```
# Create a bank instance
my_bank = Bank("MyBank")

# Create customer instances
customer1 = Customer("Alice", "123456")
customer2 = Customer("Bob", "789012")

# Add customers to the bank
my_bank.add_customer(customer1)
my_bank.add_customer(customer2)

# Perform transactions and associate with customers
customer1.deposit(1000)
customer2.deposit(500)
customer1.withdraw(300)

# Print bank information, customer information, and transactions
print(my_bank)
print(my_bank.get_customer("123456"))
print(my_bank.get_customer("789012"))

transaction1 = Transaction(customer1.account_number, "Withdrawal",
300)
transaction2 = Transaction(customer2.account_number, "Deposit",
500)

print(transaction1)
print(transaction2)
```

Step 3:

Test the program by performing transactions and retrieving customer information using the bank's methods. Ensure that transactions are associated with the correct customers.

Step 4:

To conclude the task, provide a summary of the bank's customers and their account details by printing the bank's string representation and the customer objects' string representations.

Part 2 End:

Congratulations! You have successfully extended the Object-Oriented Python program to create a simple banking system that manages multiple customers and their accounts. You can now view the bank's information and customer details.