OBJECT-ORIENTED PROGRAMMING (OOP) TASK IN PYTHON - PART 1

Objective:

In this two-part task, you will create an Object-Oriented Python program that models a simple banking system. In Part 1, you will design the core classes for customer accounts and transactions.

Instructions:

Step 1:

Create a Python class named `Customer` that represents a bank customer. The `Customer` class should have the following attributes and methods:

Attributes:

- `name` (str): The customer's name.
- `account_number` (str): A unique account number for the customer.
- `balance` (float): The current account balance.

Methods:

- `__init__(self, name, account_number)`: The constructor method that initializes the `name`, `account_number`, and sets the `balance` to 0.0.
- `deposit(self, amount)`: A method that allows the customer to deposit a specified `amount` into their account. Update the account balance accordingly.
- `withdraw(self, amount)`: A method that allows the customer to withdraw a specified `amount` from their account, provided they have sufficient balance. Update the account balance accordingly.
- `get_balance(self)`: A method that returns the current account balance.
- `__str__(self)`: A method that returns a string representation of the customer's information in the format "Name: [name], Account Number: [account_number], Balance: [balance]".

Step 2:

Create a Python class named `Transaction` that represents a bank transaction. The `Transaction` class should have the following attributes and methods:

Attributes:

- `transaction_id` (int): A unique identifier for the transaction.

- `account_number` (str): The account number associated with the transaction.
- `transaction_type` (str): The type of transaction (e.g., "Deposit" or "Withdrawal").
- `amount` (float): The transaction amount.
- `timestamp` (str): The timestamp of the transaction.

Methods:

- `__init__(self, account_number, transaction_type, amount)`: The constructor method that initializes the `account_number`, `transaction_type`, `amount`, and generates a unique `transaction_id` and timestamp.
- `__str__(self)`: A method that returns a string representation of the transaction's information in the format "Transaction ID: [transaction_id], Account Number: [account_number], Type: [transaction_type], Amount: [amount], Timestamp: [timestamp]".

Step 3:

Create instances of the `Customer` class and perform transactions using the `Transaction` class. Here's an example:

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

# Perform transactions
customer1.deposit(1000)
customer2.deposit(500)
customer1.withdraw(300)

# Print customer information and transactions
print(customer1)
print(customer2)

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

print(transaction1)
print(transaction2)
```

Part 1 End:

Congratulations! You have successfully designed the core classes for customer accounts and transactions. In Part 2, you will create additional functionality to manage multiple customers and their accounts within a simple banking system.