7.2C

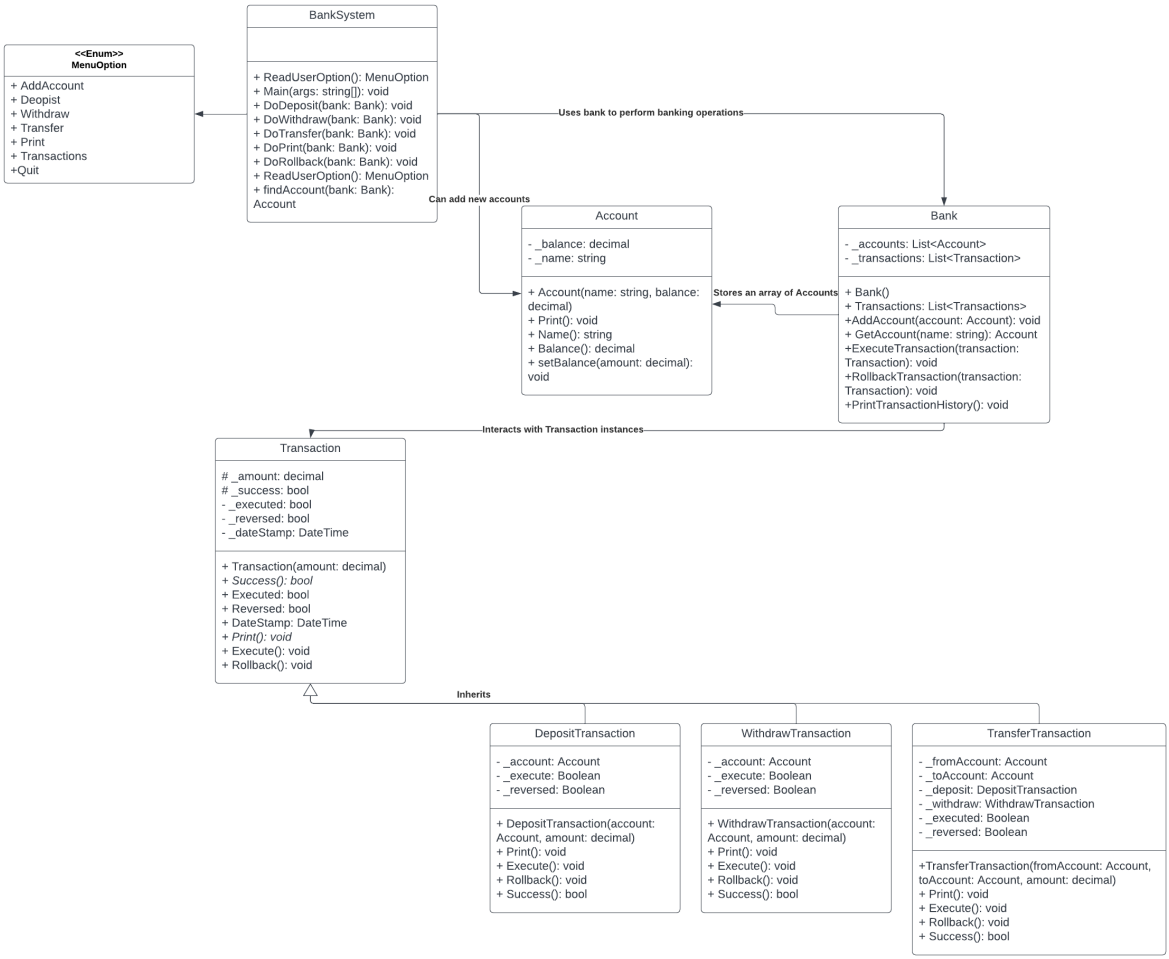
While working on 7.1P I have learnt to OOP concepts of abstraction, encapsulation, inheritance and polymorphism, and have implemented them in my finished code.

Abstraction is symbolised in the UML diagram with *italics,* abstraction involves simplifying complex classes by focusing on the essential methods and properties of a class while hiding the unnecessary details. In the Transaction class, we can see that it is determined as abstract. It also holds abstract methods such as Print() and Success(). This means that any class inheriting the Transaction class must provide its own implementation of these methods.

Encapsulation involves bundling a class's data and methods. Encapsulation restricts direct access to some of the object's components, providing a controlled access and maintaining consistency of the object's state. For example, in the Account class, both instance variables _balance and _name are private. However, through the public methods Balance() and Name() provide controlled access to these variables.

Inheritance involves the use of parent and child classes, where the child classes inherit properties and behaviours of parent classes. Inheritance prevents code duplication by allowing code reusability. An example of this is how the DepositTransaction, WithdrawTransaction and TransferTransaction classes are all child classes of the Transaction class, therefore inheriting Print(), Execute() and Rollback() methods.

Polymorphism is the ability of an object to take many forms. polymorphism helps us to select the specific class (either the child class or the parent class itself) at the runtime, and call the methods associated with the selected class. Thus, it allows us to perform a single action in different ways. An example of this is the Print() method in the Transaction class, since it is an abstract method, all of its child classes must have their own implementation of this method, meaning that each child class of Transaction has a Print() method. Polymorphism is used in the ExecuteTransaction() method in the Bank class as the code transaction.Print(); allows us to treat it as a Transaction object instead of individual objects.


<<Enum>>
MenuOption
+ AddAccount
+ Deopist
+ Withdraw
+ Transfer
+ Print
+ Transactions
+Quit
BankSystem
+ ReadUserOption(): MenuOption
+ Main(args: string[]): void
+ DoDeposit(bank: Bank): void
+ DoWithdraw(bank: Bank): void
+ DoTransfer(bank: Bank): void
+ DoPrint(bank: Bank): void
+ DoRollback(bank: Bank): void
+ ReadUserOption(): MenuOption
+ findAccount(bank: Bank): Account
Uses bank to perform banking operations
Can add new accounts
Account
- _balance: decimal
- _name: string
+ Account(name: string, balance: decimal)
+ Print(): void
+ Name(): string
+ Balance(): decimal
+ setBalance(amount: decimal): void
Stores an array of Accounts
Bank
- _accounts: List<Account>
- _transactions: List<Transaction>
+ Bank()
+ Transactions: List<Transactions>
+AddAccount(account: Account): void
+ GetAccount(name: string): Account
+ExecuteTransaction(transaction: Transaction): void
+RollbackTransaction(transaction: Transaction): void
+PrintTransactionHistory(): void
Interacts with Transaction instances
Transaction
_amount: decimal
_success: bool
- _executed: bool
- _reversed: bool
- _dateStamp: DateTime
+ Transaction(amount: decimal)
+ Success(): bool
+ Executed: bool
+ Reversed: bool
+ DateStamp: DateTime
+ Print(): void
+ Execute(): void
+ Rollback(): void
Inherits
DepositTransaction
- _account: Account
- _execute: Boolean
- _reversed: Boolean
+ DepositTransaction(account: Account, amount: decimal)
+ Print(): void
+ Execute(): void
+ Rollback(): void
+ Success(): bool
WithdrawTransaction
- _account: Account
- _execute: Boolean
- _reversed: Boolean
+ WithdrawTransaction(account: Account, amount: decimal)
+ Print(): void
+ Execute(): void
+ Rollback(): void
+ Success(): bool
TransferTransaction
- _fromAccount: Account
- _toAccount: Account
- _deposit: DepositTransaction
- _withdraw: WithdrawTransaction
- _executed: Boolean
- _reversed: Boolean
+TransferTransaction(fromAccount: Account, toAccount: Account, amount: decimal)
+ Print(): void
+ Execute(): void
+ Rollback(): void
+ Success(): bool