ONLINE BANKING SERVICE

Online Banking service is a project written in java using 4 different design patterns. This helped me in improving my knowledge in Object Oriented development. Dividing project into its functional areas, implementing them with their supporting design patterns and merging them into one in the last is quite hard in the early stage. Now after completing the project, it gave me confidence in using design patterns. Design patterns has made my work so easy and understandable to me. In this project the bank service provider asks for validation while entering in to check whether he is a valid user or not and then it displays the options for user to perform. These include view account details, check balance, credit, debit etc. In this project four design patterns are used. They are Composite, State, Visitor and Singleton design patterns.

Singleton design pattern is used for validation. Singleton pattern restricts the instantiation of a class to one object. To enhance and strengthen security for a user account and to provide a global point of access, a customer is assigned with a single user name and password. Here “user” is used as userID and “pass” as password. Here validation class has only one instance and provides a global point of access to it.

Composite design pattern is used to associate different accounts with complex and simple objects. An Account Interface is created to implement composite design pattern. The main thing in this interface is a accept() method. This accept() method is added to allow the visitor access to the various concrete Account classes. Each Account could be composed of other Accounts too. In the composite pattern the client treats both the primitive and composite structure uniformly which makes the client code simple. Also, adding new types of accounts is easy and the whole structure of the client need not be changed accordingly. But the design can sometimes become overly simple.

As Visitor design pattern is a behavioral pattern that enables the addition of different types of services to the Account class, where the composite pattern has been used. Various functions like checkbalance(), credit(amount), debit(amount) can be operated on different accounts. Concrete visitor classes are created for every functionality. Each concrete class has a visit function having a specific account object as parameter. This pattern allows adding new services for account classes without making any changes to the specific account classes. There are three types of visitors BalanceVisitor, CreditVisitor and DebitVisitor that visits all types of accounts. All these visitors implement the Visitor interface. Client is the interface created here to maintain connection between composite account interface and abstract visitor class.

State design pattern is used in Credit Account where all the operations like makepayment(), withdraw(), balance() and all other are performed using states. This include HasCreditLimit(), NoCreditLimit(), PaidState and ReachedCreditLimit() states. When the user has credit limit in his account, then he can perform operations. Otherwise, user can't perform these operations.

Challenges faced and how I rectified them:

In the process I have faced some challenges like picking the right design pattern, writing the code, calling a design pattern and merging them. First, when I was said to pick design patterns for my project, I went across lots of surfing, analyzing, modifying and finally selecting them. Initially I have thought of some other design patterns, but while writing code to it I was so unhappy seeing the process. so, I differed my design patterns many times and sticked to Composite, State, Visitor and Singleton design patterns. Later while writing the code, I have created classes, sub classes, concrete classes, interfaces, methods and more. I had run the individual design pattern at a time to know whether that particular design pattern is executed correctly or not.

Sometimes I made mistakes in creating objects for the classes and calling them at correct place. but after running them individually I corrected my errors and I thought it was done. Then came the toughest part, writing the main class, where I have to Merge all the design patterns I have written. Creating objects and calling then in one class has made me think of my project twice and finally after many rectified errors it has executed. Sometimes the correct logic wouldn't run and gave exceptional errors. At that time, I studied a lot about many things in the internet in writing same logic in different ways. I came to know about how the same logic can be wrong in some cases and what are the modifications that needs to be done in order to correct them. I have not only learned about object-oriented programming but also java, eclipse and GitHub. I was very new to eclipse and GitHub in the first week but now I learned about them, their functionalities and how to work in them. I am also very new to writing unit test cases, though I haven't learned about how to write them fully, I tried writing them and implemented two unit test cases.