Banking System using Design patterns

January 26, 2014

Contents

1 Introduction

Banking is a dynamic field which is well integrated with Computer Science and has evolved alongside it. The banking domain is characterized by transaction processing. Banking transactions in its simplest form represent events such as customer withdrawing or depositing money. Transaction processing implies the update of customer's accounts using the transactions that occurred with respect to individual accounts. Such updates are done at the instance of the transaction. The Banking system manages the customer's accounts, their personal details, annual income and records of old and new transactions. Here comes the usage of the design patterns into the system created for this project.

Design patterns are expressive, reusable and provide a common vocabulary of software solutions that can express large solutions-succinctly. They provide a readymade solution that can be adapted to different problems as necessary. In relation to object models, they bring reuse and consistency to the entire Object-Oriented software development process. The visitor and composite design patterns have been defined and used to implement the system functionalities on the developed banking system and not only make the software more agile but also better structured so that it conforms to the Software Engineering development standards. Its goal main goal is to eliminate the manual clerical job as the software does almost all of the work.

On this project three of the four patterns that were reviewed in on the paper were implemented, those patters are which are composite, visitor and singleton. The composite pattern is a structural pattern that allows the clients to treat individual objects and compositions of objects homogeneously. It arranges the objects into tree structures. In this system, this pattern is associated with the different types of Accounts which can be complex or simple objects. For example, cash account is a complex object and checking/savings account is a simple object. Similarly, credit account is complex and Visa/ MasterCard account is simple. The visit() functions are used as the common functions of both simple and complex account objects.

The visitor 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. In this application, services are the various functions that can be operated on different types of accounts like checkbalance(), credit(amount), debit(amount) and statement(). 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.

The singleton pattern is a design pattern that restricts the Instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system. The concept is sometimes generalized to systems that operate more efficiently when only one object exists, or that restrict the instantiation to a certain number of objects. This pattern is utilized to make sure that every client that connects to the database uses the same instance of the database, instead of creating a new instance for each client that connects to their account.

To build this system the Java SE 7 programming language and version were used, alongside the Eclipse IDE version Kepler. The Java Swing GUI widget toolkit was used to create the graphical user interface for the program and local database on Microsoft Access 2010 was implemented so that the program could extract the data from there.

The theoretical part of design patterns which included things such as the different types of categories for patterns, types of patterns, how they work and small snippets of code to show how was learned in class. With this project the goal was to take the theoretical part learned in class and implement it in a real world case, to see how these patterns work and interact in a real system. Not only implement them but also do design choices such as analyzing and choosing which patterns are suitable for the system and in what areas will they help. It is great practice so that we can be prepared for when it's our turn to go to the field and use these tools ourselves.

1.1 Methodology

This project employs the object oriented approach of software engineering using appropriate (UML TOOLS) object oriented approach. Some of the steps to be taken are:

- Feasibility Study:
  Understanding and identifying of existing banking system and associated study of the design patterns that can be used to develop such a system.
- Analysis:
  Proper analysis and building a banking system using the feasible design patterns will be considered.
- Design:
  Designing a banking system will be achieved through an object oriented software tools (UML).
- Coding:
  This is implemented as a 3-tier architecture model using Java language. The 3 layers of the application are as follows:
  1. Java User-Interface(Swing) – Front end
  2. Java SE - Middle end
  3. MS-Access database - Back-end

1.2 Software Usage and System Requirements

- Hardware Requirements
  - 512 MB RAM or more
  - Windows XP, 7 or upgraded versions
- Programming Languages and Environment
  - JDK 1.6 or 1.6 (Development kit)
  - Java SE (Development)
  - Eclipse (IDE)
  - MS-Access (Database)

    - JDBC (Connectivity)

- Backup Media
    - Hard Disk

This software is developed using Windows-XP as system software and is tested to be executed on all the operating systems given above.

2 Implementation

The functionality of banking system is very broad and complex. Thus, many design patterns can be used to implement such a universal and comprehensive system. Our project deals with the implementation of Visitor and Composite patterns.

2.1 General Overview

2.1.1 Composite Pattern

The composite pattern is a structural pattern that allows the clients to treat individual objects and compositions of objects homogeneously. It arranges the objects into tree structures. In the banking system, this pattern can be associated with the different types of Accounts which can be complex or simple objects. 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. For example, cash account is a complex object and checking/savings account is a simple object. Similarly, credit account is complex and Visa/ MasterCard account is simple. The common functions of both simple and complex account objects is visit().

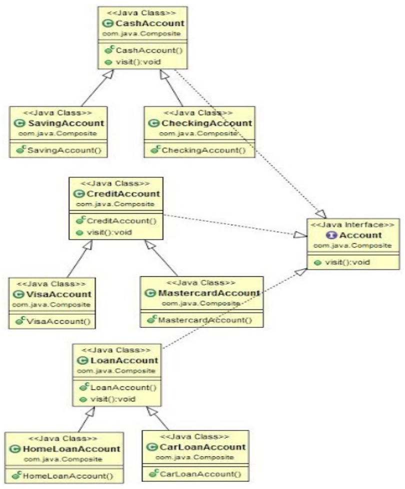
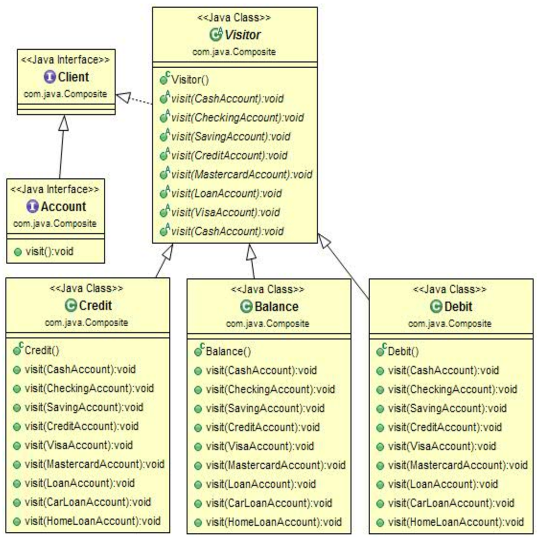


Figure 1: UML diagram of composite pattern

2.1.2 Visitor Pattern

Visitor 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. In this application, services are the various functions that can be operated on different types of accounts like checkbalance(), credit(amount), debit(amount) and statement(). 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.



<<Java Class>>
Visitor
com.java.Composite
Visitor()
visit(CashAccount):void
visit(CheckingAccount):void
visit(SavingAccount):void
visit(CreditAccount):void
visit(MastercardAccount):void
visit(LoanAccount):void
visit(VisaAccount):void
visit(CashAccount):void
<<Java Interface>>
Client
com.java.Composite
<<Java Interface>>
Account
com.java.Composite
visit():void
<<Java Class>>
Credit
com.java.Composite
Credit()
visit(CashAccount):void
visit(CheckingAccount):void
visit(SavingAccount):void
visit(CreditAccount):void
visit(VisaAccount):void
visit(MastercardAccount):void
visit(LoanAccount):void
visit(CarLoanAccount):void
visit(HomeLoanAccount):void
<<Java Class>>
Balance
com.java.Composite
Balance()
visit(CashAccount):void
visit(CheckingAccount):void
visit(SavingAccount):void
visit(CreditAccount):void
visit(VisaAccount):void
visit(MastercardAccount):void
visit(LoanAccount):void
visit(CarLoanAccount):void
visit(HomeLoanAccount):void
<<Java Class>>
Debit
com.java.Composite
Debit()
visit(CashAccount):void
visit(CheckingAccount):void
visit(SavingAccount):void
visit(CreditAccount):void
visit(VisaAccount):void
visit(MastercardAccount):void
visit(LoanAccount):void
visit(CarLoanAccount):void
visit(HomeLoanAccount):void

Figure 2: UML diagram of Visitor pattern

2.1.3 Singleton Pattern

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. A bank account can be owned by multiple authorized customers. The bank must ensure that an account is accessed by only one authorized customer at a particular duration of time.
The singleton pattern has been used in the Account class to implement this pattern. In order to maintain the integrity in the database which has been created and connected before the login screen appears to the user singleton pattern has been used in the project. This avoids the duplicates in the system and ensures that the database gets updated accordingly.

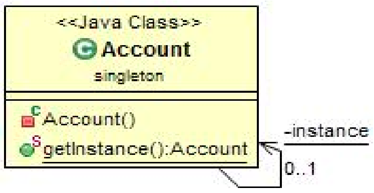


Figure 3: UML diagram of Singleton pattern

2.2 Interfaces and Classes

2.2.1 Account

Account interface has been used in implementation of composite pattern.

- An interface Account has been created whose functionality is to add an account and accept any new Visitor.
- First, a general interface is created for Account object. 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 would be composed of other Accounts too.
- Implementation of this interface provides a way to contain other Accounts for example CashAccount consists of both SavingAccount and CheckingAccount.

2.2.2 Visitor

Accounts used in banking system such as CashAccount, CreditAccount creditAccount, CheckingAccount, SavingsAccount, VisaAccount and MasterAccount implements the Account interface with a common accept() method.

There are three types of visitors BalanceVisitor, CreditVisitor and DebitVisitor that visits all types of accounts. All these visitors implement the Visitor interface. As the name specifies:

- BalanceVisitor - contains the specific visit method that returns the balance of a particular account type.
  1. It contains the visit() method to get balance for all the accounts.
  2. For Composite Accounts such as CashAccount, the visit( ) method returns sum of the Balances of its child Accounts by visiting each of its child Account. For example in case of CashAccount it returns the sum of Balances of CheckingAccount and SavingsAccount.
  3. For Simple Accounts such as CheckingAccount the visit() method returns the Balance of CheckingAccount.
- CreditVisitor - contains the specific visit method that deposits the money to a particular account type.
  1. It contains the visit() method to deposit certain amount to all the accounts.

2. For Accounts such as CheckingAccount the visit() method adds the specified amount of money to the existing Balance of CheckingAccount.
3. Any change in credit Account is reflected in the composite Account for example if certain amount is credited to CheckingAccount, it is reflected in CashAccount as well.

- DebitVisitor - contains the specific visit method that withdraws the money of a particular account type.

  1. It contains the visit() method to withdraw certain amount from all the accounts.
  2. For Accounts such as CheckingAccount the visit() method withdraws the specified amount of money from the existing Balance of CheckingAccount. Any change in Debit Account is reflected in the composite Account for example if certain amount is withdrawn from CheckingAccount, it is reflected in CashAccount as well.

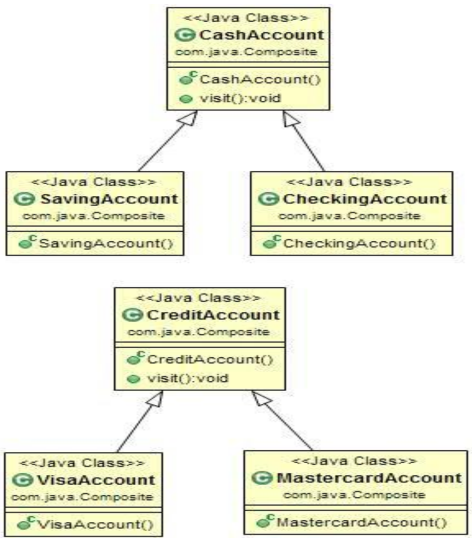


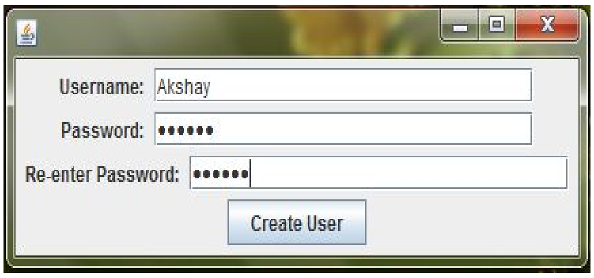
Figure 4: UML diagram of scenario

3 Results

The Banking System is developed using Java and MS-Access fully meets the objectives of the system for which it has been developed. This work has three design patterns implemented and the system has reached a steady state where all bugs have been eliminated. The system is operated at a high level of efficiency and security and all the bankers and users associated with the system understand its advantage and exploit them accordingly.The project's banking system is similar to the one used by the bankers and customers. The projects has implementations of several administrative modules such as:

3.1 SignUp Frame

This frame has been designed for the new customer to login to the system by creating authentic login-in details using the information provided by the bank. It's GUI has been shown below.



Username: Akshay

Password: ••••••

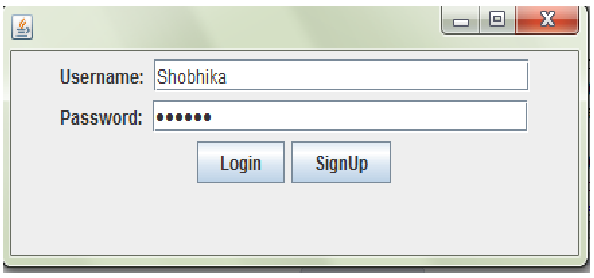
Re-enter Password: ••••••

Figure 5: Sign-up Frame

Using this frame the user creates a valid account to enter the banking system and the details will be updated in the bank records.

3.2 Login Frame

This frame acts as the initial interface between the client and the banking system. Initially, the system verifies the credentials entered by the client and then allows complete access. If the user is new he will be allowed only after creating valid user inforamtion. The figure below shows the login frame of the banking system.



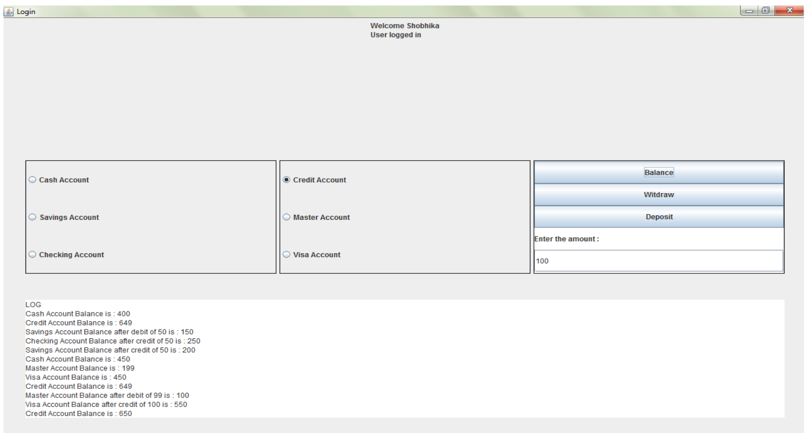
Username: Shobhika

Password: ••••••

Figure 6: Login Frame

3.3 HomePage Frame

The frame shows the complete set of information as the user is logged into the system. It has various accounts to choose from such as Cash/Savings/Checking/Credit/Master/Visa. Also, additonally it displays the cash withdrawal/deposit using the button implementation. The figure below shows the UI for the Home Frame.

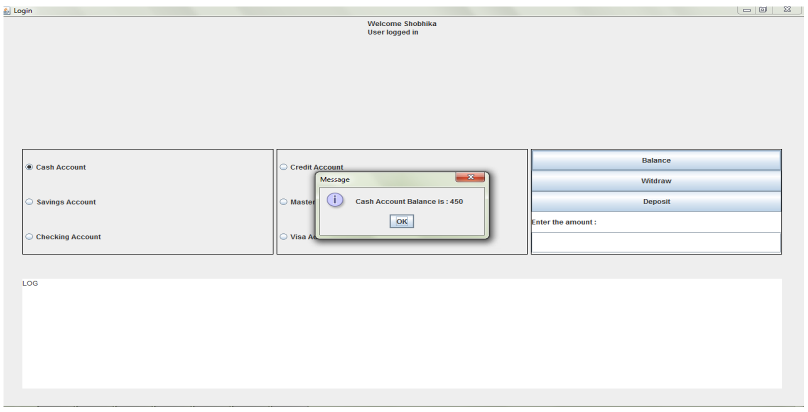


```
LOG
Cash Account Balance is : 400
Credit Account Balance is : 649
Savings Account Balance after debit of 50 is : 150
Checking Account Balance after credit of 50 is : 250
Savings Account Balance after credit of 50 is : 200
Cash Account Balance is : 450
Master Account Balance is : 199
Visa Account Balance is : 450
Credit Account Balance is : 649
Master Account Balance after debit of 99 is : 100
Visa Account Balance after credit of 100 is : 550
Credit Account Balance is : 650
```

Figure 7: Homepage Frame

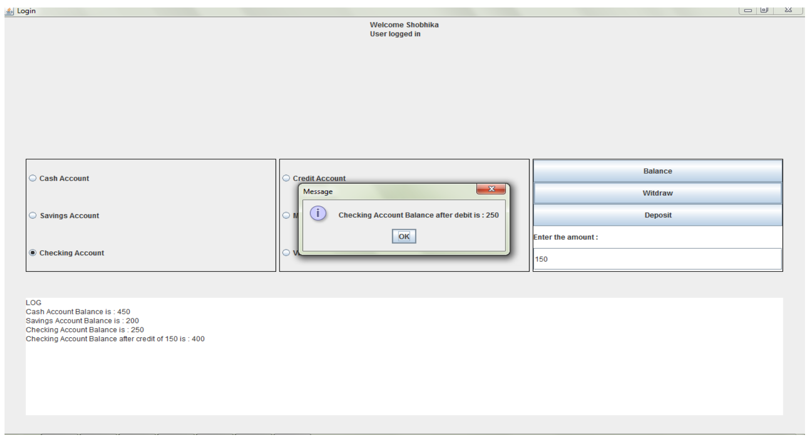
3.4 Cash/Checking Account Frame

These frames indicate the account balances after the transaction has been made in the banking system. Additionally, a pop-up screen/frame has been implemented to show the current balance in the selected account each time a transaction has been completed.



Cash Account Balance is : 450

Figure 8: Cash/Checking Accounts Frame



Welcome Shobhika
User logged in

(●) Checking Account

Message: Checking Account Balance after debit is : 250

Enter the amount : 150

```
LOG
Cash Account Balance is : 450
Savings Account Balance is : 200
Checking Account Balance is : 250
Checking Account Balance after credit of 150 is : 400
```

Figure 9: Cash/Checking Accounts Frame

4 Conclusion

Today's world is a global village run by paperless systems. Thus more technologies and innovations will still evolve which will make the cashless transactions easily accessible and affordable. Firstly, banking services are quite varied and one of its best features is putting the user in control. This form of system can also be efficiently developed using the design patterns and it is intended to solve any requirement specification. Reuse and redesign should always be part of the culture of software development organizations. Developing reusable components can often simplify design.

Our banking system can be extended to an On-line banking system giving the user increased accessibility to their account information. Users of online banking service scan access their account information, bill payments, transfers and investments from anywhere in the world.

Without any doubt, the international scope of banking system provides new growth perspectives and business is a catalyst for new technologies and new business processes. With rapid advances in telecommunication systems and digital technology, banking system has become a strategic weapon for banks to remain profitable. It has been transformed beyond what anyone could have foreseen 25 years ago.

5 Future Scope

From analyzing this project build there are some things that could be added or changed for a future build of this system, in other words the future scope will be:

- Include the other design pattern discussed on the paper that was submitted previously (façade).
- The database would be transferred from a local one (MS-Access) to an online one (Oracle). The reason an Oracle database was picked is because of

the great connectivity and support that it has with the Java programming language.

- Add more functions to the visitor pattern such as transfer funds.
- Other pattern implementations that we might have missed on the original paper and during this project but up until now.
- Add the functionality for a client to be able to pay bank loans directly on the system.
- Include an account statement for each account that the client has.
- Add a functionality to pay bills directly from the account.

6 Bibliography and References

1. M.Hemalatha , A.Vikneshraj, N. Adithya Prasanna, Sri Manakula Vinayagar .“Identification and Implementation of Design Patterns in Mobile Banking”. International Journal of Computers and Technology. [Online]. Vol. 6 (Issue 2). Available: http://cirworld.com/index.php/ijct/article/view/1325.
2. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object Oriented Software. Addison-Wesley Professional, 1994, pp 1-345
3. Enoh John Enoh, “Resuable Object-Oriented Patterns in Banking Applications”, Internet: http://www.scribd.com/doc/886861/Reusable-ObjectOriented-Patterns-in-Banking-Applications, Dec. 11, 2007.
4. ”Design Patterns”, Internet: http://en.wikipedia.org/wiki/Design_Patterns.
5. ”Design Patterns using JAVA”, Internet: http://www.javacamp.org/designPattern/, Jun. 2013 [Nov.26 2013].
6. Steve Almasy,”Internet Tranforms Modern Life”, Internet:http://www.cnn.com/2005/TECH/internet/0 Oct. 10, 2005 [Nov. 4, 2013]

7 Appendix

The link to the banking system developed through this project has been given below. It contains all the classes, interfaces and UML diagrams developed for the banking system.

https://docs.google.com/file/d/0B6qD-QDaxQuna3JZN0ZCa0VJMVk/edit?pli=1