

CS 319 - Object-Oriented Software Engineering  
Final Report

E-Banking System

Group 2-11

Tuna Seyis

Kübra Şen

Kaan Kale

Table Of Contents

1. Introduction.......................................................................................4

2. Requirement Analysis.......................................................................4

2.1. Overview..............................................................................................4

2.2. Functional Requirements.....................................................................5

2.3. Non-Functional Requirements.............................................................5

2.4. Constraints...........................................................................................5

2.5. Scenarios..............................................................................................6

2.6. Use Case Models.................................................................................7

2.7. User Interface.....................................................................................11

3. Analysis..........................................................................................................12

3.1. Object Model.....................................................................................12

3.1.1. Domain Lexicon...................................................................12

3.1.2. Class Diagrams.....................................................................13

3.2. Dynamic Models...............................................................................12

3.2.1. State Chart............................................................................15

3.2.2. Sequence Diagrams..............................................................16

4. Design ………………………………………………………………………22

4.1 Design Goals………………………………………………………..22

4.2 Subsystem Decomposition …………………………………………23

4.3 Architectural Patterns ………………………………………….......24

4.4 Hardware/Software Mapping ………………………………………25

4.5 Addressing Key Concerns …………………………………………25

4.5.1 Persistent Data Management ………………………………25

4.5.2 Access Control and Security ………………………………25

4.5.3 Global Software Control …………………………………..26

4.5.4 Boundary Conditions ………………………………………26

5. Object Design ………………………………………………………………27

5.1 Pattern Applications ………………………………………………...27

5.1.1 Façade Pattern ……………………………………………...27

5.1.2 Bridge Pattern ………………………………….…………..27

5.2 Specifying Contracts ………………………………………………..28

6. Conclusion......................................................................................................30

7. References.......................................................................................................32

1. **Introduction**

E-Banking System is a basic software program which we decided to develop. The main purpose of this program is making banking operations easier, faster and more useful than other banking programs for our users.

In this program our customers can open saving account or checking account, apply to take loan and get reply from the server whether it accepted or denied, credit money, debit money, transfer money to other customers who also uses our bank and users can see their operations which they did before.

The program will work on desktop and internet connection mandatory. Users can control it by mouse and fill fields with their keyboard.

1. **Requirements**
   1. **Overview**

To use the program for first time the user must be sign up for our e-banking

system with their own information after this they encounter with login page and with their ID’s and password user can easily access to main menu. In this menu user will see six different choices. These are “Create Account”, “Take loan”, “Debit Money”, “Credit Money”, “Transfer Money”, “List Operations”. In first choice which is “Create Account” user can open a new “Saving Account” or “Checking Account”. When “Taking Loan” selected user enters amount of money and progress of their payments in number of months. In “Debit Money” and “Credit Money” user can draw money from their accounts or put money to their accounts respectively. When “Transfer Money” button pressed user can send money to other users however they also must use our program to succeed. Last button is “List Operations” and when this clicked users can see their last transactions which they did before.

**2.2.** **Functional Requirements**

1. Users will be able to create an account (saving/checking) via system.

2. Users will be able to credit money to their accounts and debit money from their accounts.

3. Users can transfer money from their accounts to other accounts.

4. Users can take a loan and choose its type with program.

5. System will make loan transactions automatically from indebted user’s account.

**2.3. Non-Functional Requirements**

1. User interface will be designed in order to be easy to use by all end users. So, minimalistic and simple design will be used. All operation controls should be simple and identifiable.

2. System should be secure. In order to provide that user passwords will be stored encrypted.

3. Program should be time-efficient so users can do most of the operations at most 3 clicks.

**2.4. Constraints**

1. Program will be implemented in C++

2. User interface will be implemented by WinAPI for C++

**2.5. Scenarios**

**1. Opening First Accounts**

After getting his first paycheck Ahmet wants to open a bank account. He opens the program and sign up and creates a membership to the bank. After that he logins with his information and from menu he chooses open an account from the menu. After that he enters his information and credits his paycheck to account. After couple of months when he got bonus from his company, he wants to keep his money from a saving account. Firstly he creates his saving account and picks an account type. After that he credits his bonus to his account and transfers it to a saving account. After that he withdraws (debit) small sum of money for his daily expenses.

**2. Getting Loan**

After 2 years Ahmet get bored from using public transport and wants to buy a car. But his money is not enough for it. So he opens his account and goes to loans segment. He enters needed amount and payment time. After he got confirmation from bank he withdraws his money and buys a new car. During payment time his debt will transacted directly from his checking account.

**3. Sending money**

When Ahmet able to support himself completely with his income. He wants to support his family and wants to send money to his home. He calls his parents and asks their account number. After receiving their number Ahmet opens the program and goes to transfer segment. He enters his parents account number and transfer amount and clicks the confirmation button.

**4. Paying loans early**

After his 5th year at the company Ahmet receives a promotion and thanks to his efforts he get huge bonus from his company. With that money Ahmet wants to pay his loan early. He opens the program, goes to loan segment and chooses pay option. In new segment he enter remaining amount of the debt to text field and hits the confirmation button. Amount will be deducted from his checking account.

**2.6. Use Case Models**

This section provides information about the main use case model of Banking System. Main actor is user. User can sign up the system then can log in. After that he/she see the main menu and can choose the one of the options which are in the use case diagram. After the all these steps, operation checking step in and inform the user about whether operation is done or not.

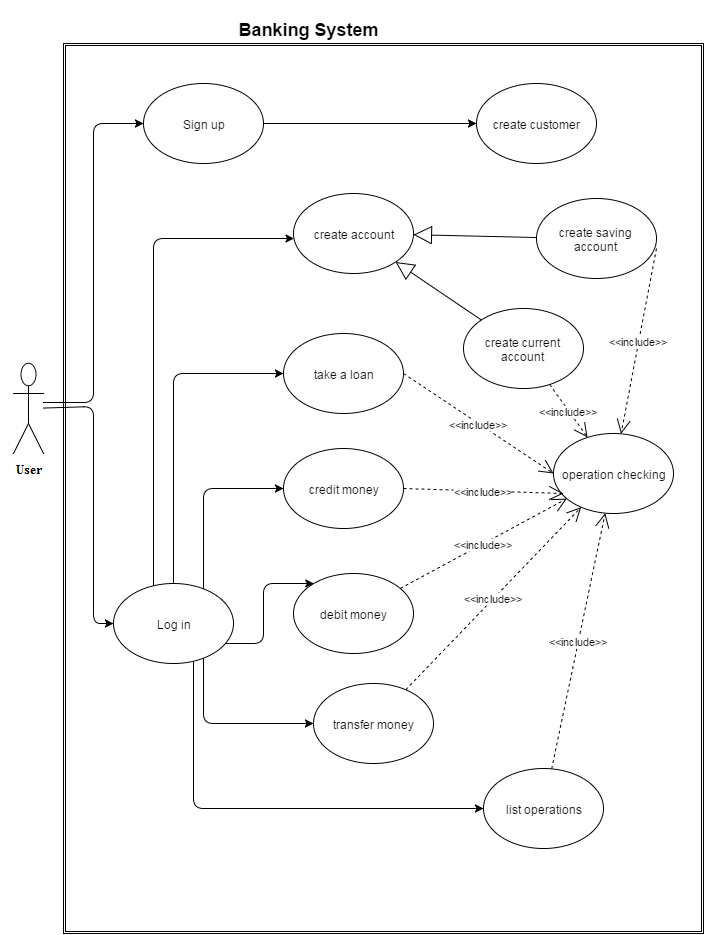


Figure 1 use case diagram

Some detailed use case explanations:

**Use Case Name: Sign up**

**Primary Actor**: User  
**Stakeholders and Interests:**  
-User wants to sign up the banking System  
-System ask the information about user  
-System wants user to determine a password  
  
**Pre-conditions:** -  
**Post-condition: -**  
  
**Entry Condition:** User selects “Sign up” from Start menu.  
**Exit Condition:** User selects “OK” and return Start menu.

**Success Scenario Event Flow:**  
  
1.System accept the user.  
  
**Alternative Flows:**  
A. If user desires to return start menu at any time:  
 A.1. User selects “Return to Start Menu” button to return start menu.  
 A.2. System displays start Menu.

B. System does not accept the user:

B.1. System report that there is another person with the same phone number  
 B.2. User selects another phone number

**Use Case Name: Log in**  
  
**Primary Actor**: User  
**Stakeholders and Interests:**  
-User wants to log in the Banking System  
-System ask user information to the user  
  
**Pre-conditions:** User registered before  
**Post-condition: -**  
  
**Entry Condition:** User selects “Log in” from Start Menu.  
**Exit Condition:** User selects “Back” and return Start Menu.

**Success Scenario Event Flow:**  
1.User Log in the System and transferred to the main menu  
  
**Alternative Flows:**  
A. If user desires to return start menu at any time:  
 A.1. User selects “Return to Start Menu” button to return Start menu.  
 A.2. System displays Start Menu.

B. if the log in information is wrong

B.1. System report that there is wrong information  
 B.2.a User write the right information  
 B.2.b User return the start menu

**Use Case Name: Transfer Money**  
  
**Primary Actor**: User  
**Stakeholders and Interests:**  
-User choose the transfer money option in the main menu  
-System ask which account user wants to transfer from  
- System ask which account user wants to transfer to  
- System ask the amount of money which user wants to transfer  
  
  
**Pre-conditions:** account that user wants to transfer from has the determined amount of money  
**Post-condition: -**  
  
**Entry Condition:** User selects “Transfer Money” from main menu.  
**Exit Condition: -**User selects “back” and return main menu.  
 -Money transferred and successful page turned over the main menu

**Success Scenario Event Flow:**  
  
1.System transfer the money  
  
**Alternative Flows:**  
A. If user desires to return main menu at any time:  
 A.1. User selects “Back” button to return main menu.  
 A.2. System displays main Menu.

Account dos not have the amount of money:

B.1. System report that the money is not enough   
B.2.a User selects another account  
B.2.b User return to the main menu

**2.7. User Interface**

User firstly sees the start menu. This includes sign up and log in option. After the log in, there is a main menu consist of the option list ; Take Loan, Create Account, Credit Money, Debit Money, Transfer Money, List operation and return.

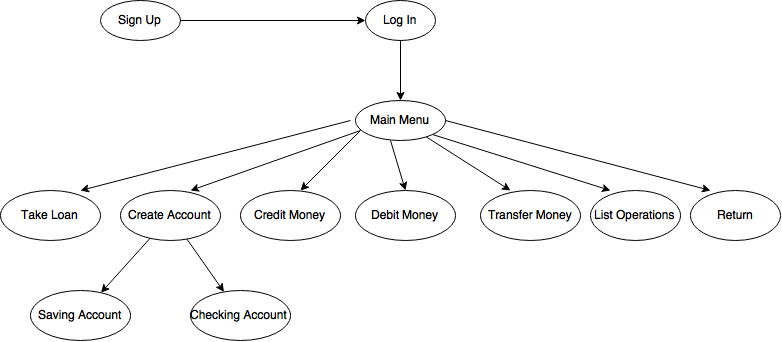
****

Figure 2 : User interface

**3. Analysis**

**3.1. Object Model**

**3.1.1. Domain Lexicon**

We mingle with some banking term. Explanation of the term which we use in the project is below

Loan: taking the money from bank for certain time. In this operation bank transfer money to user’s account and provide them payment plan.

Credit Money: putting the money to a certain account. All users can credit money to their own account

Debit Money: taking money from a certain account. All users can take money from their own account

Transfer Money: transferring the money from a one account to another. All user can transfer money from their own account to any other account

Saving Account: accounts maintained by retail financial institutions that pay interest but cannot be used directly as money in the narrow sense of a medium of exchange.

Checking Account: A transactional deposit account held at a financial institution that allows for withdrawals and deposits

**3.1.2. Class Diagram**    
 There are 7 classes in the Project. User Menu class is to organize main menu and start menu. Bank class is the main class in the code. İt include main function of the bank system. Account class is parent class of saving account and current account and has a relation with user class.

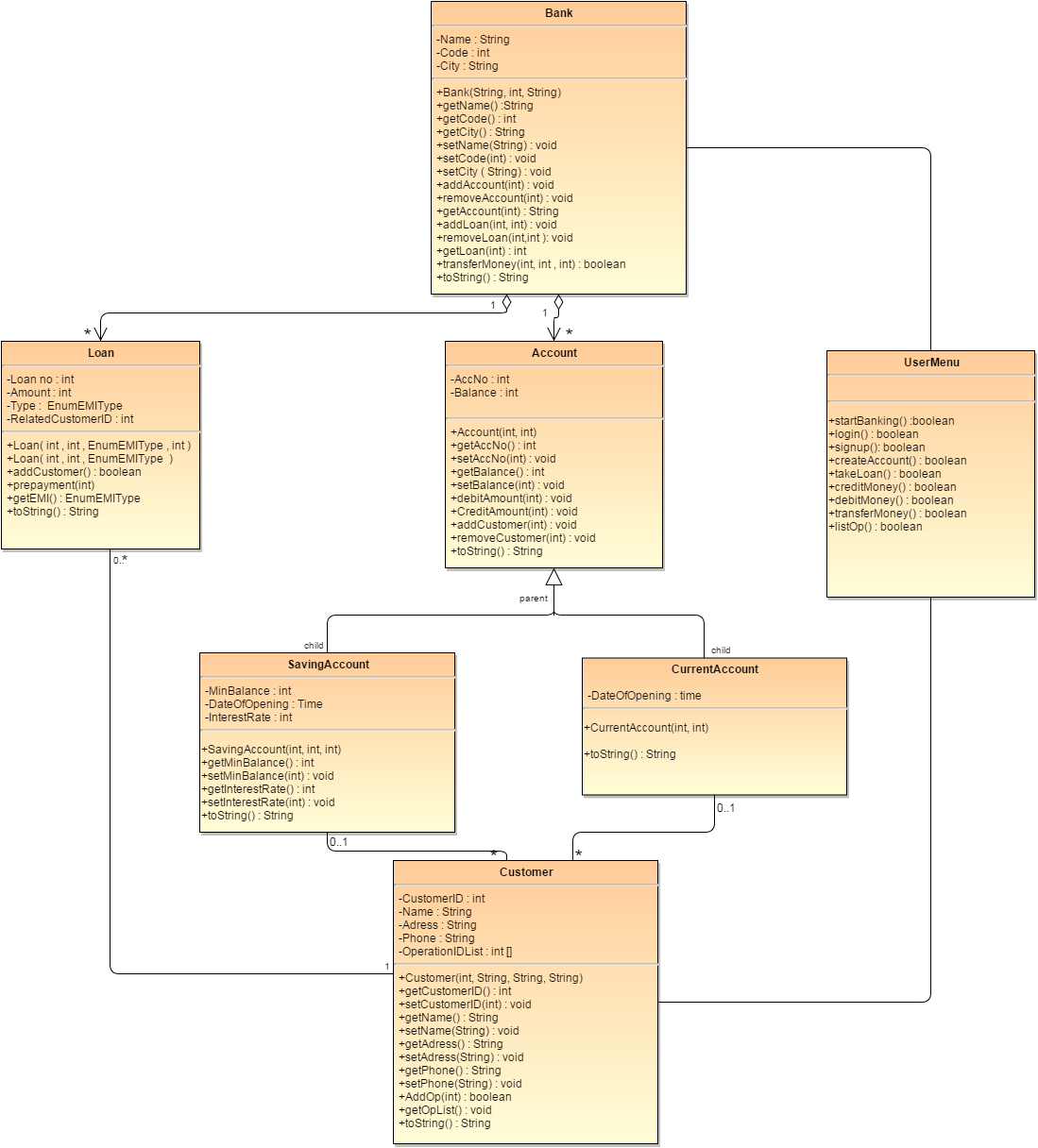
****

Figure 3 : Class diagram

**3.2.1. State Chart**

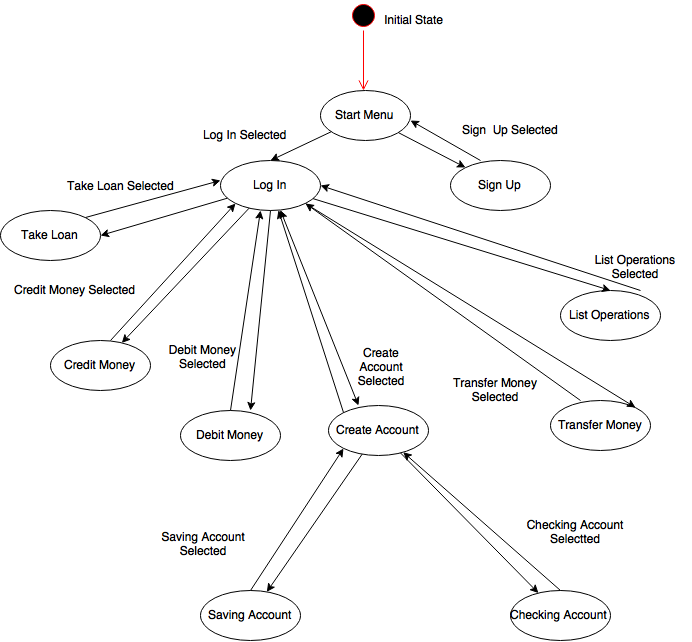


Figure 4 : State Chart Diagram

**3.2.2 Sequence Diagrams**

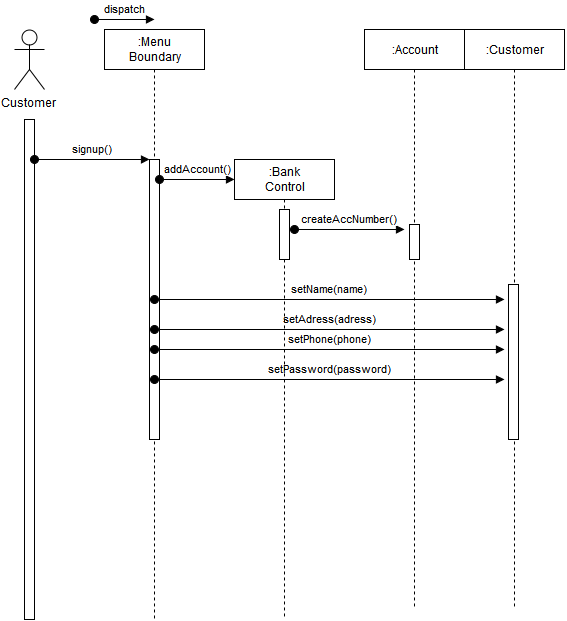


Figure 5 : Sequence Diagram 1

When user chooses creating new membership to the bank, bank automatically creates account number for the user and asks for user name, address, phone and password. Program stores the information, encrypts the password and returns a confirmation message to user.

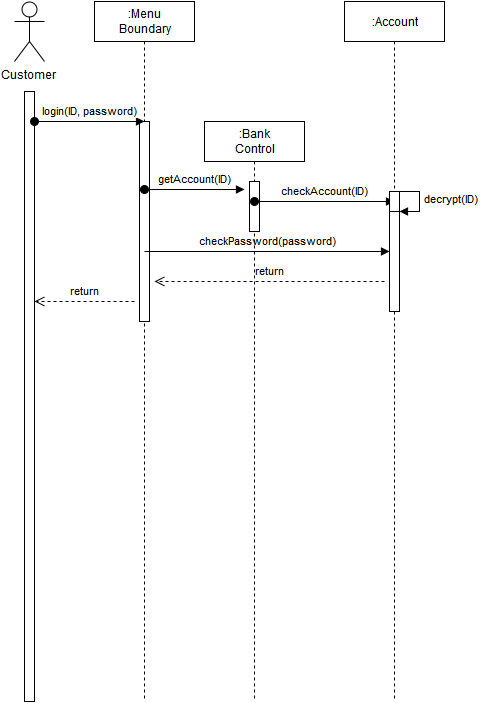


Figure 6 : Sequence Diagram 2

User enters ID and password through the menu and control unit checks ID and password matches with password in storage and if passwords checks with each other, main menu will show up or an error message will show up.

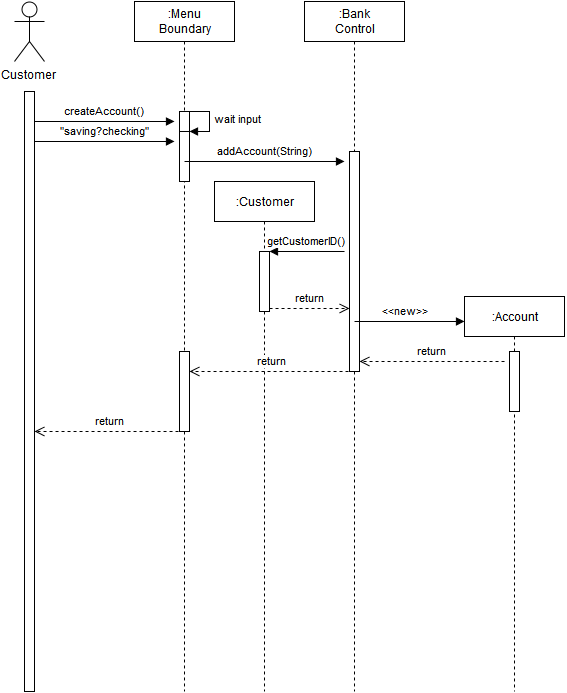


Figure 7 : Sequence Diagram 3

When a user choose create an account option, menu will ask to user type of the account. After that according the type of the account, program will create new account object with the selected type and bind that account to user’s account ID. New account initially created empty.

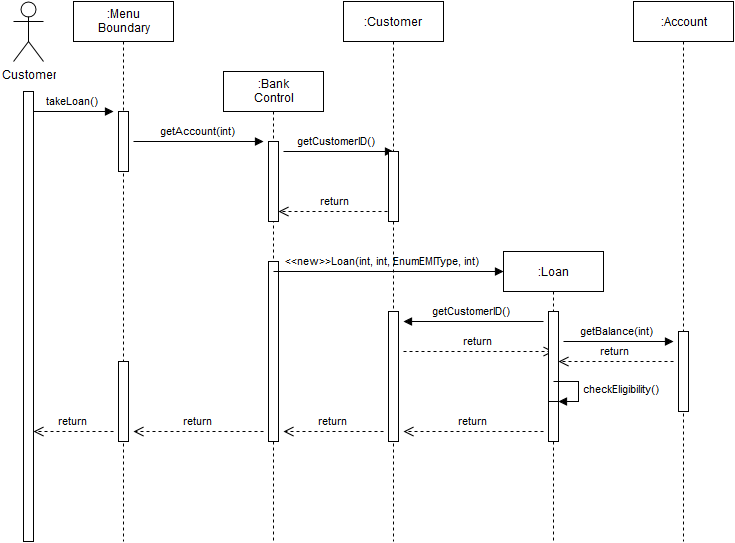


Figure 8 Sequence Diagram 4

When a user wants to take a loan from the bank, control unit will find account number of the user, then creates a new loan object and ask user loan amount and amount of installments. After user entered desired amount of money and installment program calculates total debt and check eligibility user according to the user account.

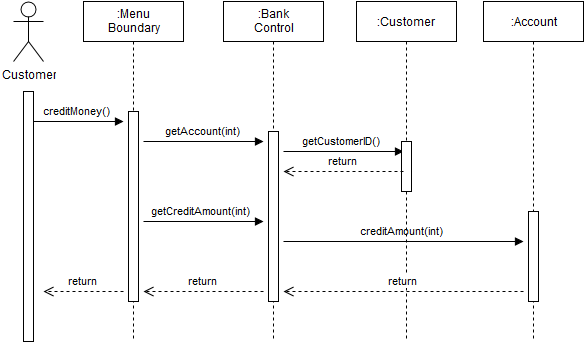


Figure 9 : Sequence Diagram 5

When users choose credit money to own account, control unit gets the user’s account then adds the entered amount.

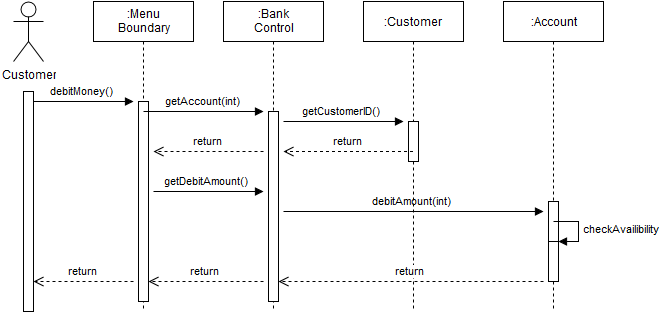


Figure 10 : Sequence Diagram 6

When users choose debit money, program brings the account of the user and checks availability of entered money. If money is available in the account money will be given.

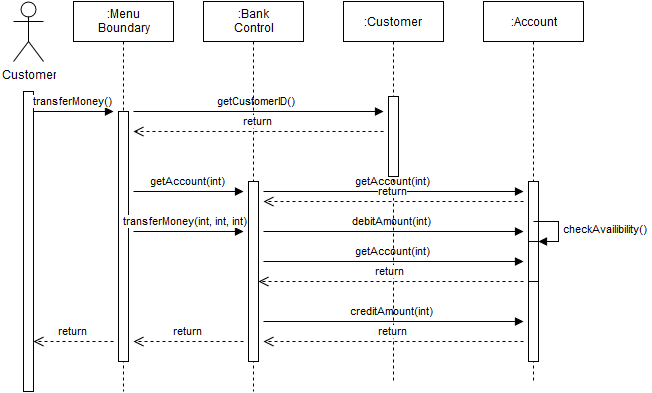


Figure 11 : Sequence Diagram 7

If user wants to transfer money to another account, control unit bring the account of the user and debits the entered amount from user’s account and credits the same amount of money to desired account.

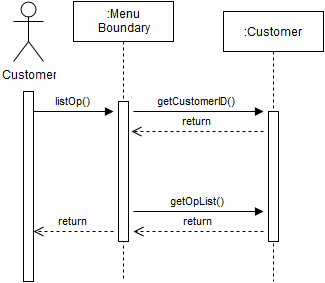


Figure 12 : Sequence Diagram 8

When an user wants to see previous list of operations, control unit brings up the previous operations list.

**4. Design**

**4.1 Design goals**

**4.1.1 Easy to use**

The main goal of our system is to create a system that is easy to use for user. People who use the system should not need to any additional information and carry out the process line taking loan, transferring Money easily. Every user who is older or younger should easily find desired operations

**4.1.2 Security**

Security is the most important part on the system. Any other people except banker and user cannot reach the account information and option menu.

**4.1.3 Functionality and performance**

Functionality and performance are also important for the system. People who use the system do not want to spent time too much and slowness of system increases the possibility of coincidence if the network problems occur.

**4.1.4 Reliability**

 Developing efficient and reliable network components required for our system. Because of the security of our clients’ money, we use reliable network components.

**4.1.5 Robustness**

Developing a robust system can handle errors such as network problems and invalid inputs and give meaningful feed-back to users. İn network problems, process should no mire down. There should not be connection problem between account and operation.

**4.1.6 Good documentation**

A good documentation is necessary for a projects design phase as it is also important in all other phases. By the help of a good documentation the implementation phase gets easier and it leads to better and more effective implementation.

**4.1.7 Availability**

We design out system with high availability. Users can use the system at 24 hous-7day

**4.1.8 Modifiability and maintainability**

Developing reusable components that are easy to modify and maintain by paying attention to low coupling- high cohesion principle. We strongly believe that, using well-known design patterns can help us to attain this goal.

**4.2 System decomposition**

The system is divided into relatively independent three parts to clarify how it is organized. Since the decisions we made in identifying subsystems will affect significant features of our software system

System is separated into three subsystems which are focusing on different cases of software system. Subsystem are user interface , system logic, storage part. All three subsystem is connected with each other and the class which is in different subsystem use other class.

User interface subsystem is consist of menu class. Menu class is only class which users can interact. Menu class use bank class and customer class in system logic subsystem. System logic subsystem is consist of the main class which do operations. Storge subsystem is include class which keep information about operation like loan or information of account

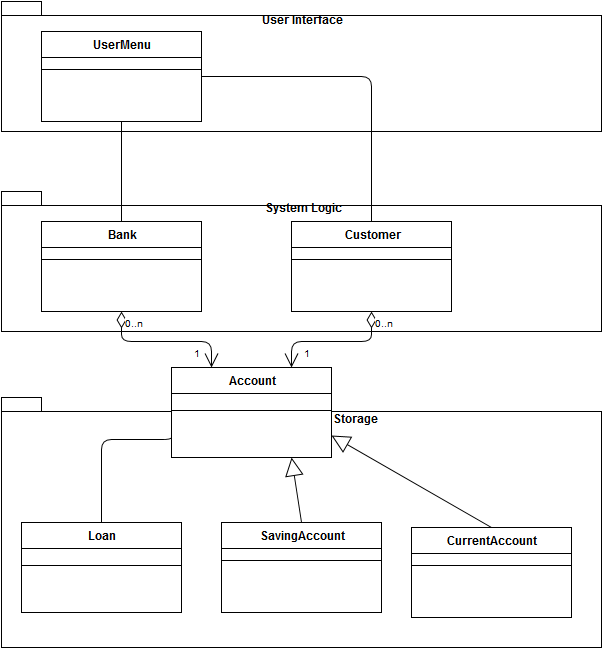


Figure 13: System Decomposition

**4.3 Architectural Patterns**

In our system we used three layers, User interface, system logic and storage. Our layer decomposition is closed architectural style and that means that every layer can only access to the layer below it. Our top layer is user interface, because it is not used by any other layer. It provides necessary operations to create a menu in order to receive user input. One layer below, we put system logic layer. In this layer system’s functions are controlled. Our bottom layer is storage, in which all information about users and accounts stored here.

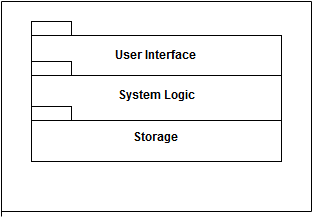


Figure 14: Layers

In our design we are also using model, view, controller system. For our bank management system we want to isolate information of the program from the user interface by using controller in between model and view. Classes grouped in user interface constitute view, classes grouped in system logic constitute controller and classes grouped in storage part constitute model of our system. In this way view communicates model only via controller and when view need modifying model stays unaffected.

**4.4 Hardware Software Mapping**

Program will be implemented in C++ programming language. As hardware configuration, our management system needs a basic keyboard for entering information and choosing menu options. System requirement will be minimal, basic computer which includes an operating system to run .exe file

Computer has to have an internet access for connecting the system.

**4.5. Addressing Key Concerns**

**4.5.1. Persistent Data Management**

Our e-banking program is a complex system and needs database system to store account information and transactions. These transactions are loan,credit money, debit(withdraw) money, transfer money.

**4.5.2. Access Control And Security**

As mentioned before (in Hardware / Software Mapping section) , our e-Banking System require network connection. An account can not be opened from more than one IP number. So there will be controls for access to accounts.

**4.5.3. Global Software Control**

We are using Model-View-Controller, so we are using event-driven mechanism because it has seperated logic, view and control parts. Our view part consists of user menu, logic part consists of bank and customer and our control part consists of account, saving account, current acount and loan.

**4.5.4. Boundary Conditions**

One customer can’t open more than one account. When a new customer wants to open new account if he/she enters an existing mail address or existing personal ID number system will give a warning.

If user enters wrong account ID or password system opens same log in page and user should enter again to open his account.

When a customer wants to transfer money, user must enter right informations to send it, if not user gets a warning and system opens main menu page.

* If entered account ID does not exist in the system user gets ”The account ID you entered does not exist! Please check again. ” error.
* If user has not enough money to send user gets a “You don’t enough money insufficient funds” warning.

**5. Object Design**

**5.1 Pattern Applications**

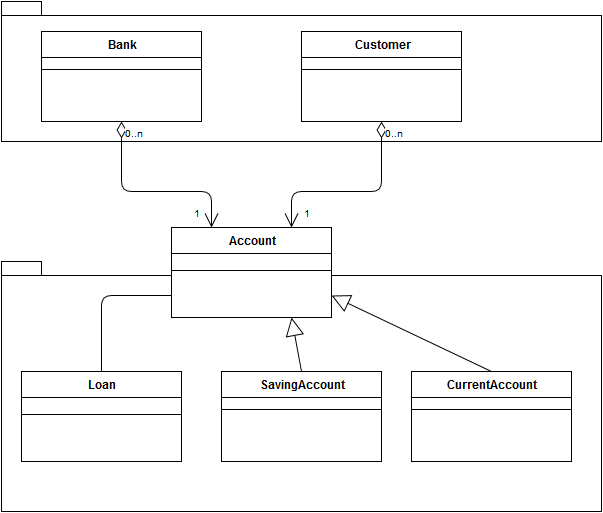


Figure 14: Patterns

**5.1.1 Facade Pattern**

In storage sub-system we used façade pattern to make it easier to use. Bank and customer classes both use account class and operate through that class. Account class gives them access to data of the accounts and loans. Account class simply a unified interface for remaining classes in the same sub-system.

**5.1.2 Bridge Pattern**

Bridge pattern in our system decouples saving and current account from account class. So that they can vary independent from each other and account class support multiple account types.

**5.2 Specifying Contracts**

**1.Context Bank:: getName() :String**

Post: result = name //returning name of the bank

**2.Context Bank:: getCode() : int**

Post : result = code //returning code of the bank

**3.Context Bank:: getCity() : String**

Post : result = city //returning city of the bank

**4.Context Bank:: setName(name : String) : void**

Post : self.name = name //changing name of the bank

**5.Context Bank:: setCode(code : int) : void**

Post : self.code =code //changing name of the bank

**6.Context Bank:: setCity ( city : String) : void**

Post : self.city = city //changing city of the bank

**7.Context Bank:: addAccount(int) : void**

Pre : menu.createAccount() //creating account is precondition of //adding account

**8.Context Bank:: addAccount(int) : void**

Post: account.Account(accountNo : int , balance : int) //creating account is //precondition of adding //account

**9.Context Bank:: transferMoney(mainAccountNo : int, targetAccountNo: int , amauntOfMoney :** **int) : boolean**

Pre :amountOfMoney >= 0 //the money in the account must more //than 0 to transfer it

**10.Context Bank:: transferMoney(mainAccountNo : int, targetAccountNo: int , amauntOfMoney :** **int) : boolean**

Pre : account.getBalance() >= amountOfMoney //account must has the amount of //money which want to transfer

**11.Context Bank:: transferMoney(mainAccountNo : int, targetAccountNo: int , amauntOfMoney :** **int) : boolean**

Post: result = true //if the operation is done, result must be true

**12.context menu :: login() : boolean**

pre: customer.isCustomer(customerId : int) //only customer can log in

**13.context menu :: createAccount() : Boolean**

pre : customer.isCustomer(customerId : int) //only customer can create //account

**14.context menu :: takeLoan() : boolean** )

pre : customer.isCustomer(customerId : int) //only customer can take loan

**15.context menu :: creditMoney() : boolean**

pre : customer.isCustomer(customerId : int) //only customer can credit money

**16.context menu :: creditMoney() : boolean**

pre : customer.account.getBalance() > 0 //to credit money, balance must //be more than 0

**17.context menu :: debitMoney() : Boolean**

pre : customer.isCustomer(customerId : int) //only customer can debit money

**18.context menu :: transferMoney() : Boolean**

pre : customer.isCustomer(customerId : int) //only customer can transfer //money

**19.context menu :: transferMoney() : Boolean**

pre : customer.account.getBalance() > 0 // to transfer money, customer //must have money

**20.context : account :: creditAmount(amountOfMoney: int)**

pre : amountOfMoney<=self.balance //to credit certain amount of //money, customer must have at //least that amount of money

**21.context customer :: getCustomerID() : int**

Post : result =customerId //returning customerId

**22.context customer :: setCustomerID(customerId : int) : void**

Post : self.customerId = customerId //changing customerId

**23.context customer :: getName() : String**

Post : result = name //returning name of customer

**24.context customer :: setName(name : String) : void**

Post : self.name = name //changing name of customer

**6. Conclusion and Lesson Learned**

The main purpose of this program is to give simpler and better service to customers than other banking systems for their operations such as creating accounts, taking loans, transferring money to other customers. Users are the main actors of our e-Banking System. Customers can easily use this program if they have connection to internet. Bank store all customer information, and their operations.

Our final report have 2 main parts, these are analysis and design part. In analysis part we described our program and explained functional, non-functional and constraints. We write different scenarios for every single operation that user can encounter, and also showed and explained these with use case models for our customers. Moreover, we generate user interfaces, class diagrams, state chart and sequence diagrams. We had formed classes, objects and relationships between these in class diagrams. In state chart, we showed actions of our system. Lastly, in sequence diagrams we mentioned all scenarios to show behaviors in every single action.

In design part, we split our system into 3 subsystems and while this decomposition phase we pay attention to low coupling and high coherence. Then, we started to design architectural patterns and determined the relationship between packages. Lastly, we explained persistent data management, access control and security, global software control and boundary conditions in our design part report.

In this project we learned very important theoretical information about software development. We understand and implement requirement analysis, system design and object design phases which must be done before development. We raised our teamwork skills and experienced it in difficult situations through this project. We learned how to make a user friendly, simple and well-designed program and applied in this project.

**7. References**

Draw.io,. 'Flow Chart Maker & Online Diagram Software'. N.p., 2015. Web. 28 Oct. 2015.