**CS442 – Software Engineering**

**Course Project**

**Final Project Report**

BANKING APPLICATION

Professor: Dr. Mark Grechanik

TA: Yanlin Han

**Group 2:**

Abhishek Kanchan

Akshay Patil

Mahesh Gadge

Neha Shah

Prasad Nair

Rose Mary George

Table of Contents

[1 Introduction 3](#_Toc387449499)

[1.1 Purpose of the system 3](#_Toc387449500)

[1.2 Scope of the system 3](#_Toc387449501)

[1.3 Objectives and overview of the project 3](#_Toc387449502)

[1.4 Definitions, acronyms, and abbreviations 4](#_Toc387449503)

[1.5 References 4](#_Toc387449504)

[2 Proposed system 5](#_Toc387449505)

[2.1 Overview 5](#_Toc387449506)

[2.2 Functional requirements 5](#_Toc387449507)

[2.2.2 Use cases 6](#_Toc387449508)

[2.2.3 Use Case Models 11](#_Toc387449509)

[2.3 Nonfunctional requirements 12](#_Toc387449510)

[2.3.1 Usability 12](#_Toc387449511)

[2.3.2 Reliability 12](#_Toc387449512)

[2.3.3 Performance 12](#_Toc387449513)

[2.3.4 Supportability 12](#_Toc387449514)

[3 Assumptions 12](#_Toc387449515)

[4 User Interface 13](#_Toc387449516)

[5 Database Diagrams 16](#_Toc387449517)

[5.1 E-R Diagram 16](#_Toc387449518)

[5.2 Physical Schema 17](#_Toc387449519)

[6 Class Diagram 18](#_Toc387449520)

[7 Sequence Diagrams 19](#_Toc387449521)

[7.1 Login and Logout 19](#_Toc387449522)

[7.2 Transfers 20](#_Toc387449523)

[7.3 Shopping Order 21](#_Toc387449524)

[8 Tools 23](#_Toc387449525)

# Introduction

## Purpose of the system

The domain of the system encompasses banking domain and a shopping portal. The purpose of this project is to create an application that would interface with banking customers to create an easy experience of using a banking system. Users may seamlessly move towards shopping through the banking module. Shopping could be made easier without the need for a separate login or linking cards.

## Scope of the system

Currently we plan to implement a Bank and a Shopping Portal. Banking system will consist of functionalities for customers which includes manage accounts, bank balance enquiry, global transfers and credit card processing. Bank balance enquiry will include transaction history with facility to check transactions for specified time period. Transfers include transfer of money between accounts of same user, different users or global wire transfers. Every bank customer will have the opportunity to have a credit card which can be used to gain reward points for all purchases. Shopping portal system will include listing of products, shopping cart where users can add and/or remove products and payment system where customer can pay by credit card, reward points or directly from bank account.  We can extend the implementation to include multiple banks and single shopping portal so that multiple banks can leverage same shopping portal facilities for their customers.

## Objectives and overview of the project

The system objectives could be to allow users to:

1. Check banking transaction history
2. Transfer money between different accounts
3. Transfer money between checking, savings and credit accounts.
4. Search products in shopping portal
5. Add or remove products from shopping cart
6. Pay for the products using multiple payment methods.

## Definitions, acronyms, and abbreviations

Users - Customers of the bank

CVV - **C**ard **V**erification **V**alue

## References

[1]Bernd Bruegge, Allen H. Dutoit, *Object Oriented Software Engineering Using UML Patterns and Java*, 3rd ed. Prentice Hall, 2012.

# Proposed system

## **Overview**

The project provides the customers of the bank, an application to use various features of the bank such as depositing or withdrawing money from various accounts, transferring funds between accounts as well as to different customers. It also provides easy access to a special shopping portal designed specifically for the customer to easily purchase products using different methods of payment and earn points for every order which can be used in future orders.

## Functional requirements

1. The user credentials at login must be authenticated before allowing access into the system.
2. A transfer of funds between accounts of the same customer must not result in any change in total amount of that customer.
3. After a transfer of funds between accounts of different customer a user must be given a confirmation message of the transfer.
4. If the user pays for a product using his credit account, then the amount for the product must be deducted from his or her available credit and reward points must be added to the credit account.
5. If the user pays for a product using his or her Reward points, then it must be reflected in his or her account and the points must be deducted accordingly.
6. If the user pays for a product using his checking or savings account, then the amount for the product must be deducted from his or her checking or savings account only.
7. User must be billed only for items in his/her cart when payment is processed.
8. Products once removed from the cart must become available for other customers for purchase.
9. Reward points are added as 1 point per $10 spent and for purchasing, 10 points would earn a $1 worth of discount.
10. An order will fail if the customer has insufficient balance in his accounts.
11. If the order contains an item that is not available, it will fail.
12. A customer can view his Order history.
13. A customer can view his Transaction history.
14. A transaction will fail to complete if the customer has insufficient balance for a transfer to a different account.

### Use cases

1. Login

|  |  |
| --- | --- |
| **Name** | Login |
| Actors | User: Customer |
| Pre-Condition | Nil |
| Scenario | 1. The application displays prompts for the username and password. The Login button is displayed as well. 2. Customer enters the username, password, and presses the Login button. 3. The application verifies the Customer’s access credentials and then displays the overview. |
| Alternate Scenario | If in step 3 the application fails to verify user’s information, the system displays an error message and allows the user to log in again. |

2. Logout

|  |  |
| --- | --- |
| **Name** | Logout |
| Actors | User: Customer |
| Pre-Condition | User must be logged into the system |
| Scenario | 1. The application allows the user to logout of the system at any point of time by clicking on the Logout link. 2. The user can exit from the application. |

1. View Account

|  |  |
| --- | --- |
| **Name** | View Account |
| Actors | User: Customer |
| Pre-Condition | User must be logged in the system |
| Scenario | 1. Application allows user to check account balance after logging in the system 2. User can choose to see transaction history for each of his/her accounts    1. Checking Account    2. Savings Account    3. Credit Account 3. For a credit account user can check reward points accumulated over time. 4. For a credit account user can check credit used and credit remaining for given month. 5. User can logout from his/her account. |
| Post-Condition | 1. Account balance is displayed to user 2. Transaction history is display to user 3. User can exit from application by clicking on logout |

1. Transfer Funds
   1. Transfer between two accounts of same Customer

|  |  |
| --- | --- |
| **Name** | Transfer between two accounts of same Customer |
| Actors | User: Customer |
| Pre-Condition | User must be logged into the system |
| Scenario | 1. The application allows the user to initiate transfers. 2. The application allows the user to initiate a transfer to another self-account. 3. The application allows the user to enter account number, amount and transfer date.    1. Amount is verified as <= Account balance    2. Transfer date is verified as greater than current date. 4. Application transfers the amount to the destination account and reduces the same amount from the source account. 5. Application allows the user to logout from the system. 6. Source account can be checking or saving account. 7. Destination account can be checking, saving or credit card account. |
| Alternate Scenario | 3.   1. If amount entered > account balance, message is displayed and user is allowed to enter details again 2. If transfer date is < current date, message is displayed and user is allowed to enter date again |
| Post-Condition | 1. Total amount in the two accounts after transfer must be equal to amount before transfer. 2. User can exit from application by clicking on logout. |

* 1. Transfer between two accounts of different Customer

|  |  |
| --- | --- |
| **Name** | Transfer between two accounts of different Customer |
| Actors | User: Customer |
| Pre-Condition | User must be logged into the system |
| Scenario | 1. The application allows the user to initiate transfers. 2. The application allows the user to initiate a transfer to another customer. 3. The application allows the user to account number, enter amount and transfer date.    1. Amount is verified as <= Account balance 4. Application transfers the amount to payee and reduces the same amount from the account. 5. Application allows the user to logout from the system. |
| Alternate Scenario | 1. If amount entered > account balance, message is displayed and user is allowed to enter details again |
| Post-Condition | 1. User can exit from application by clicking on logout |

5. Shopping

* 1. Search

|  |  |
| --- | --- |
| **Name** | Search |
| Actors | User: Customer |
| Pre-Condition | User must be logged into the system |
| Scenario | 1. User enters product name and clicks on the “Search” button. 2. The application finds the product within the database and displays the found product for the user. 3. The application allows the user to view details of the product. 4. Application allows the user to logout from the system. |
| Alternate Scenario | If in step 2, the system cannot find the product, the application will produce an error. |

* 1. Cart operations

|  |  |
| --- | --- |
| **Name** | Cart operations |
| Actors | User: Customer |
| Pre-Condition | User must be logged into the system |
| Scenario | 1. User searches for products available 2. User selects the required product and adds it to cart 3. Application saves the product in the shopping cart of user. 4. Application allows user to view the cart. 5. Application allows user to remove any item from the cart. 6. The application allows user to proceed to checkout if the cart contains any items. 7. User can exit from application by clicking on logout. |

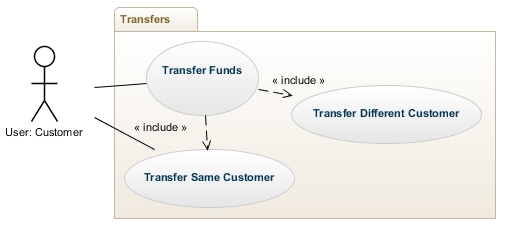
* 1. Payments

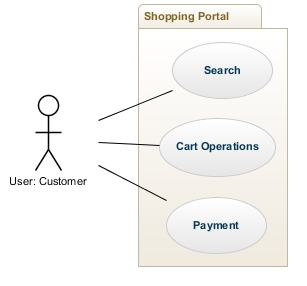
|  |  |
| --- | --- |
| **Name** | Payments |
| Actors | User: Customer |
| Pre-Condition | 1. User must be logged into the system 2. User’s cart must contain at least one item |
| Scenario | 1. The application allows the user to make a payment for the items to be purchased. 2. The application provides the user three options of payment    1. Pay by bank account only or combined with reward points. Enter the following details.       1. Amount    2. Pay by credit card only or combined with reward points. Enter the following details.       1. Credit card number       2. CVV number       3. Amount    3. Pay by reward points. 3. The application allows the user to enter the payment details. 4. Application checks for availability of the items. 5. If the items are available the application allows user to confirm the payment. 6. Once the payment is confirmed the application reflects the transaction in the user account based on choice 2.    1. User gains rewards for amount paid using credit card.    2. The reward points are deducted from the current points for amount paid using points.    3. Amount paid using account is deducted    4. Available credit is decreased by the amount paid using credit card. |
| Alternate Scenario | 1. If there is insufficient amount in any of the accounts/, insufficient points, the application would display a message and the user is not allowed to purchase the item. 2. If the items are unavailable the application displays a message and the user is not allowed to continue the purchase. 3. The user may cancel the purchase before the payment. |
| Post-Condition | 1. The items in the shopping cart must be removed based on the items purchased. 2. Quantity of the items must be updated. |

### Use Case Models

|  |  |
| --- | --- |
| **1. Authentication** | **2. Home View** |

**3. Transfers**

****

1. **Shopping Portal**  
   

## Nonfunctional requirements

### Usability

No specific training should be necessary for a user to begin using the application.

#### Graphical User Interface

The system shall provide a uniform look and feel between all the screens.

The system shall provide use of icons and toolbars.

### Reliability

1. The system shall provide storage of all user data on a database.
2. Only registered users are allowed to have access to the system.
3. Users have to authenticate each time they use the application (login).

### Performance

The product should not take more than 3 minutes for initial load.

Robustness or fault-tolerance Requirements**:**

Time to restart after failure is less than 3 minutes.

### Supportability

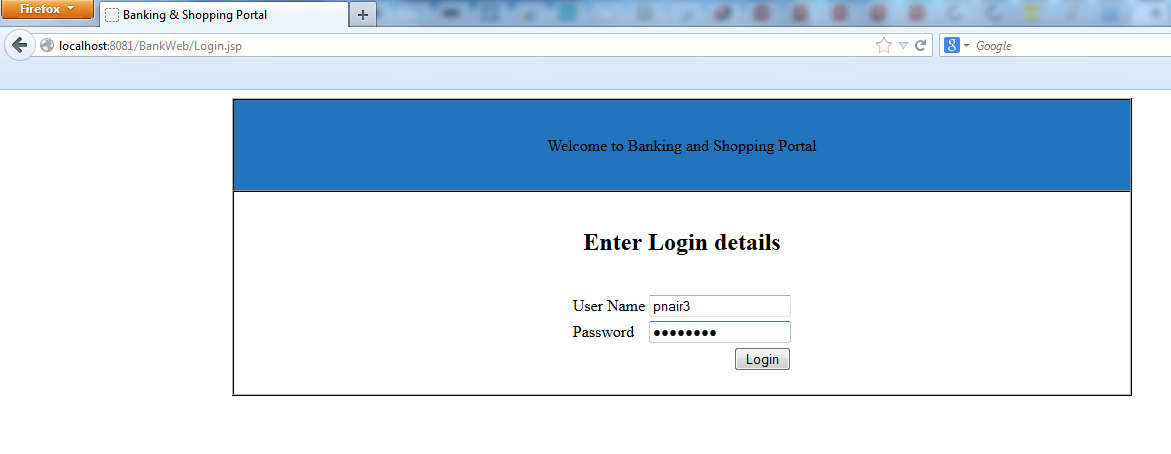
1. All program files shall include time stamps denoting authorship and date of last change.
2. The code shall be modular such that each module contains everything necessary to execute only one aspect of the desired functionality, thus allowing future changes to be made easily for anticipated updates.
3. All code for the application shall be fully documented with each function having comments for pre and post-conditions.

# Assumptions

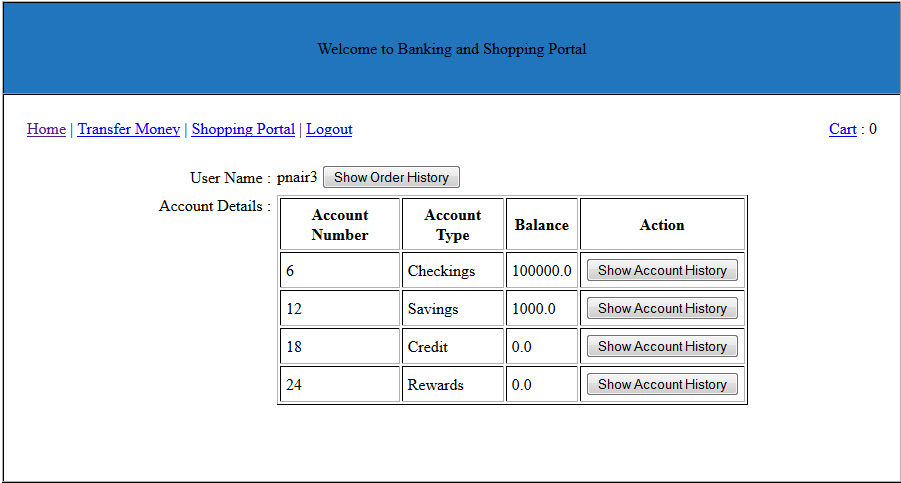
1. Account has already been created for a customer.
2. A random initial balance amount is added to the account.
3. The login user name and password is always known.
4. A credit card is pre-approved for every customer.
5. There is always a secure transfer between 2 accounts.
6. The atomicity of a transaction is always maintained.

# User Interface

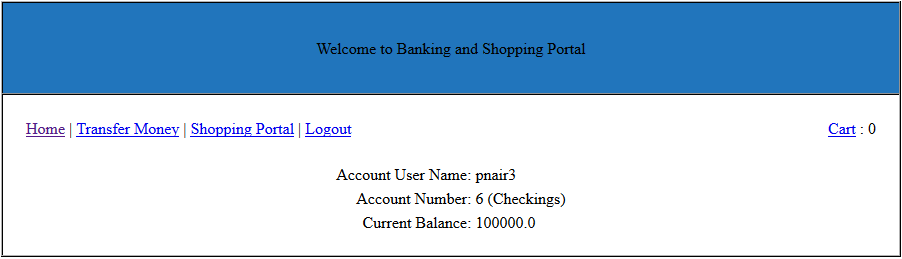
1. **Login**



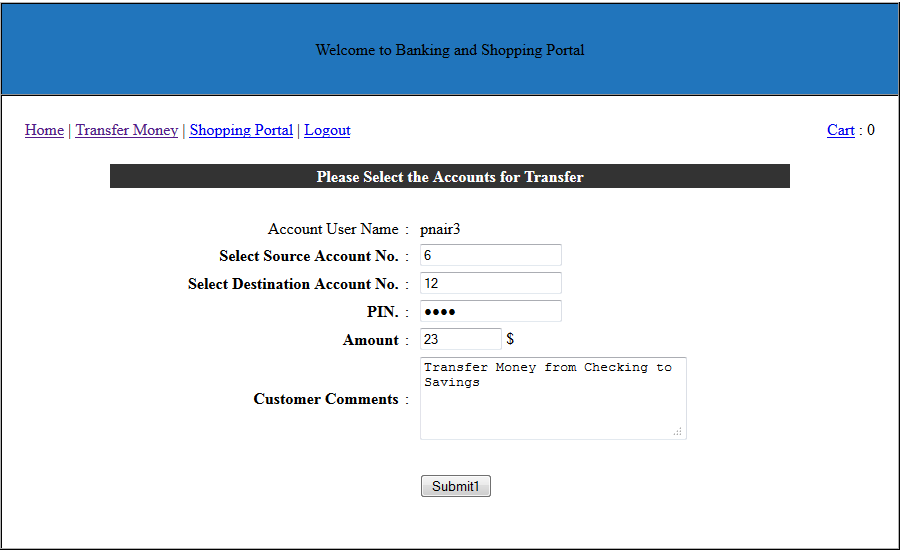
1. **Home Page**



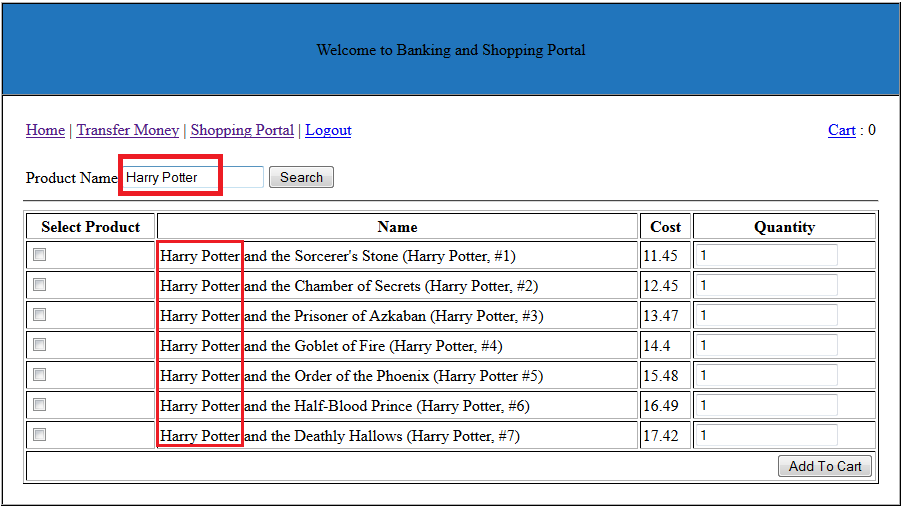
1. **Account History**



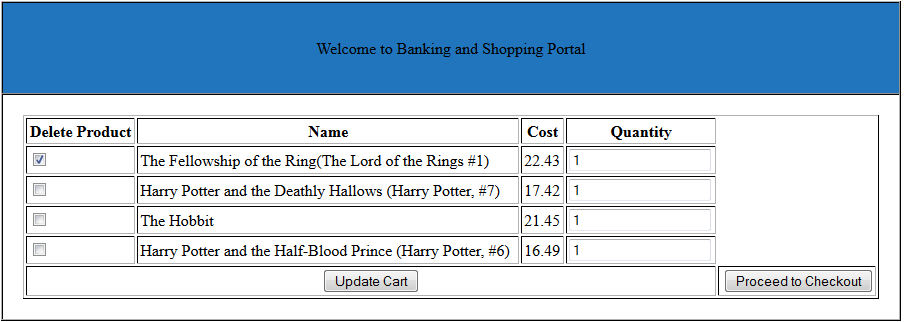
1. **Transfer Money**



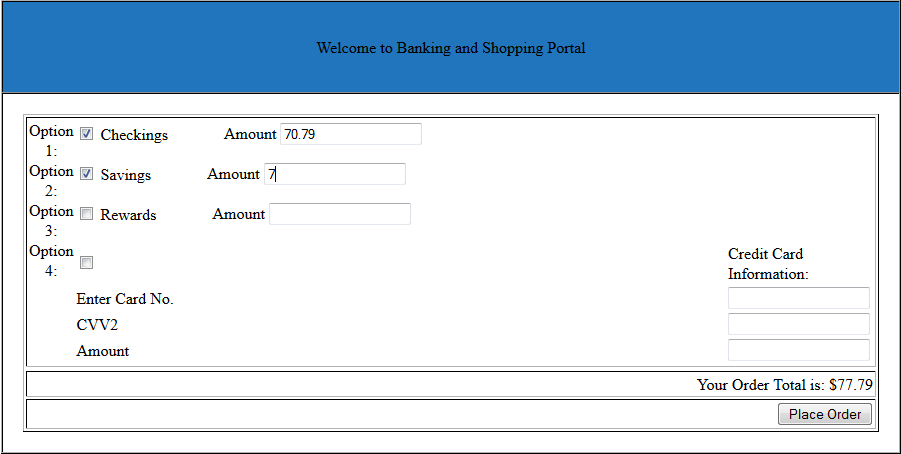
1. **Search Items**



1. **Cart**

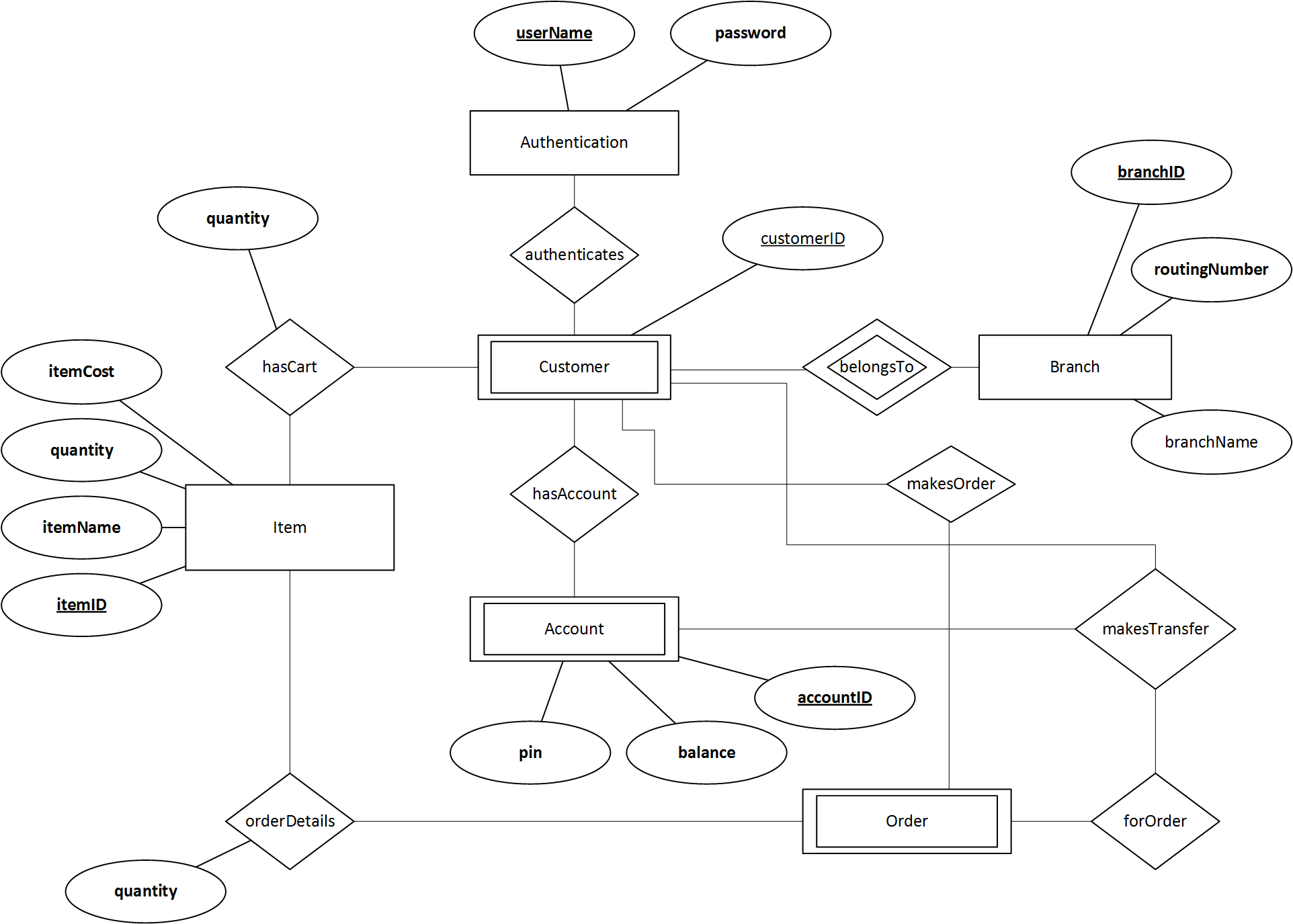


1. **Checkout**

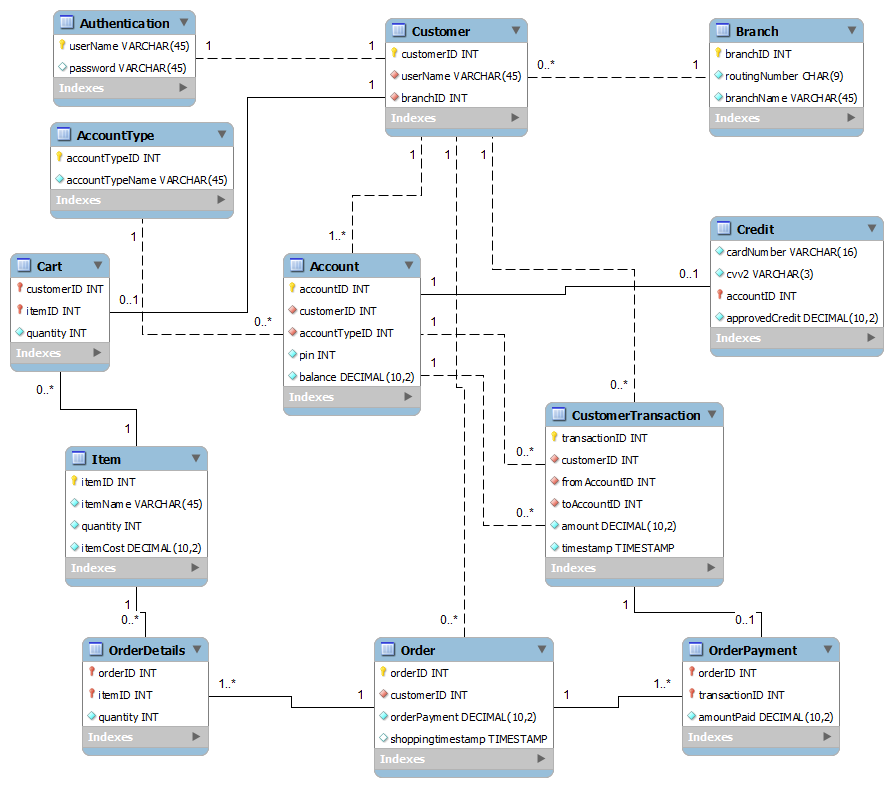


# Database Diagrams

## E-R Diagram



## Physical Schema



# Class Diagram

The class diagram is a type of the Structural UML diagram which shows the types being modified within the system. For our Banking application the class diagram shows the different class entities or “classifiers” as UML calls it and the relation between them. Once the customer logs in to the system after proper authentication, the customer object is instantiated by the 'Customer' class which has its corresponding ‘Cart’ object active till the user logs out of the system. The relation between a customer and their shopping cart is of one-one kind.

The 'Account' class represents the general banking account class which has some basic attributes which are common to most types of account. The ‘Credit’ class is a special type of account which inherits from the ‘Account’ class. The accountType attribute in the parent Account class differentiates between the different types of accounts that a user holds. The customer has a one-many relation with the different Account types. The assumption here between the customer and a type of account is that an account (AccountID) cannot have more than one holders of that account i.e Joint members of an account is not allowed in the system.

The ‘Item’ class represents the various shopping products stored in our system which has a quantity associated with it and which keeps updating as and when a user places a finalized order for these items. The relation between the user cart and the different items in the cart is of many-many type. The ‘Order’ class represents the finalized order instance which is placed by the customer from the system. It consists of one or many items of the same type or of different types and it is possible for a product (itemID) to not be placed in any existing order of the system.

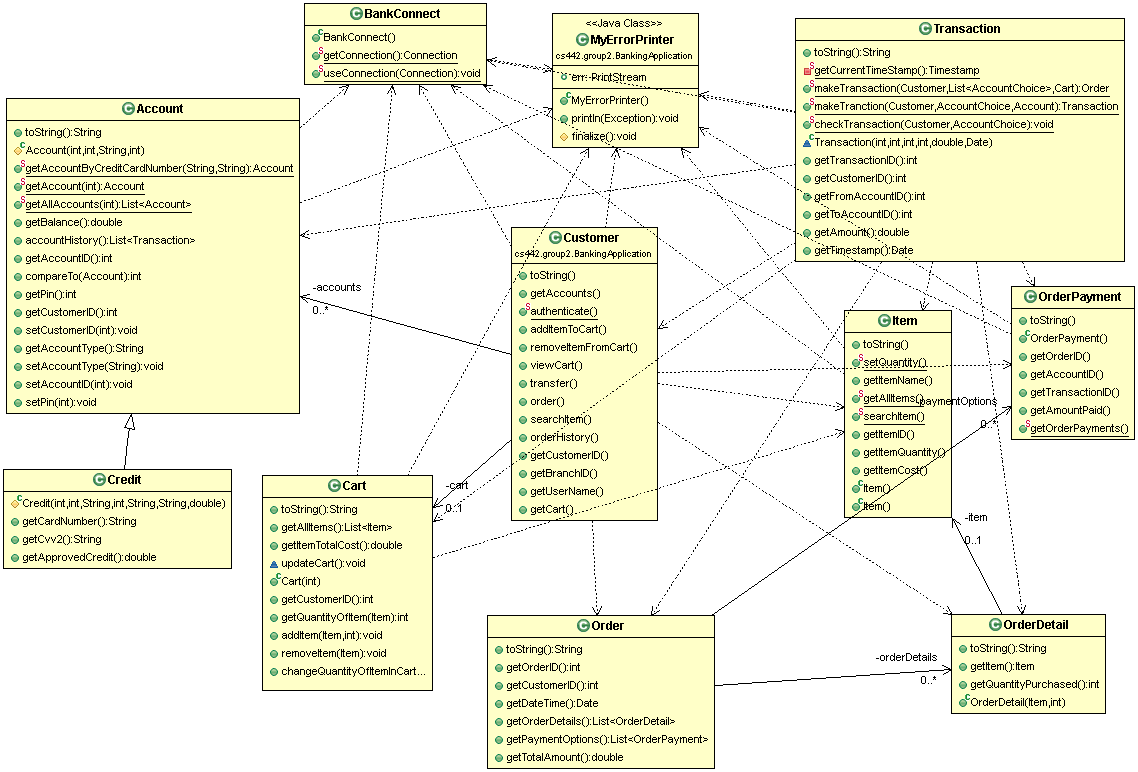
The customer has the option of making the payment through multiple accounts (savings, credit, checking, rewards), so the ‘OrderPayment’ class keeps track of the various payment options used to place a successful order by the customer. The relation between an Order and the OrderPayment class is thus of the one-many type. The ‘CustomerTransaction’ class is used to store attributes and keep records of all the successful transactions which took place in the system by various customers for the following use-cases:

1) Transfer from a customer of our system to another customer, also in our system

2) Self transfer - customer transfers money from his account to one of his other bank accounts within the system.

3) Shopping orders - customer making a payment from one or more of his accounts towards the shopping merchant for an order.

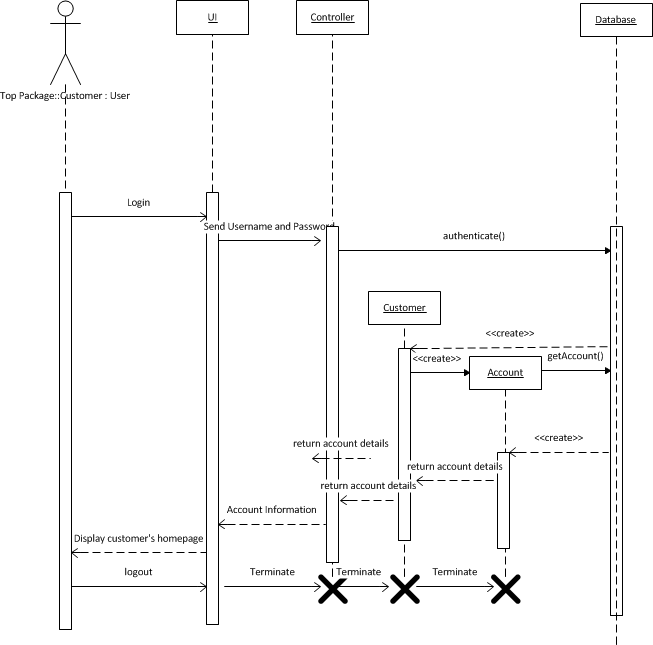
These classes represent the overall class structure of our entire system of which instances will be created during the normal functioning of the banking/shopping application.



# Sequence Diagrams

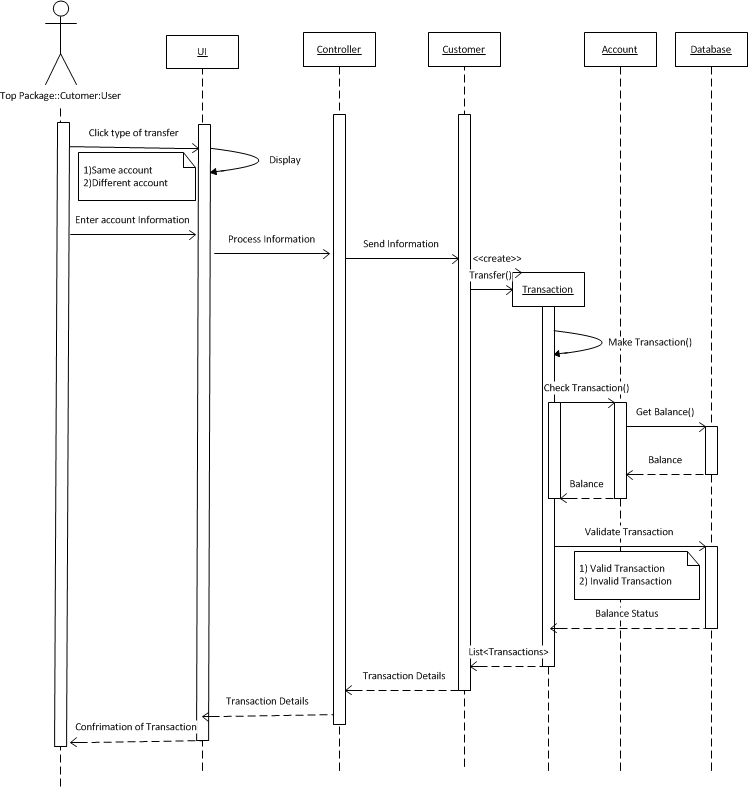
## Login and Logout

The customer enters his username and password and clicks on the Login button from the interface. The Controller then checks the customer’s credentials and matches them with the information in the database using the authenticate() method. If the credentials do not match, the database does not validate the customer and the interface displays an “error page”. In this sequence diagram, we assume that the user is a valid Customer of the Bank. Once the database verifies the above, Customer object is created and this in turn creates the Account object with the Customer’s account details. These details are returned to the controller and the interface will display the account details to the customer. If the customer decides to Logout, he/she can click on the “Logout” button.



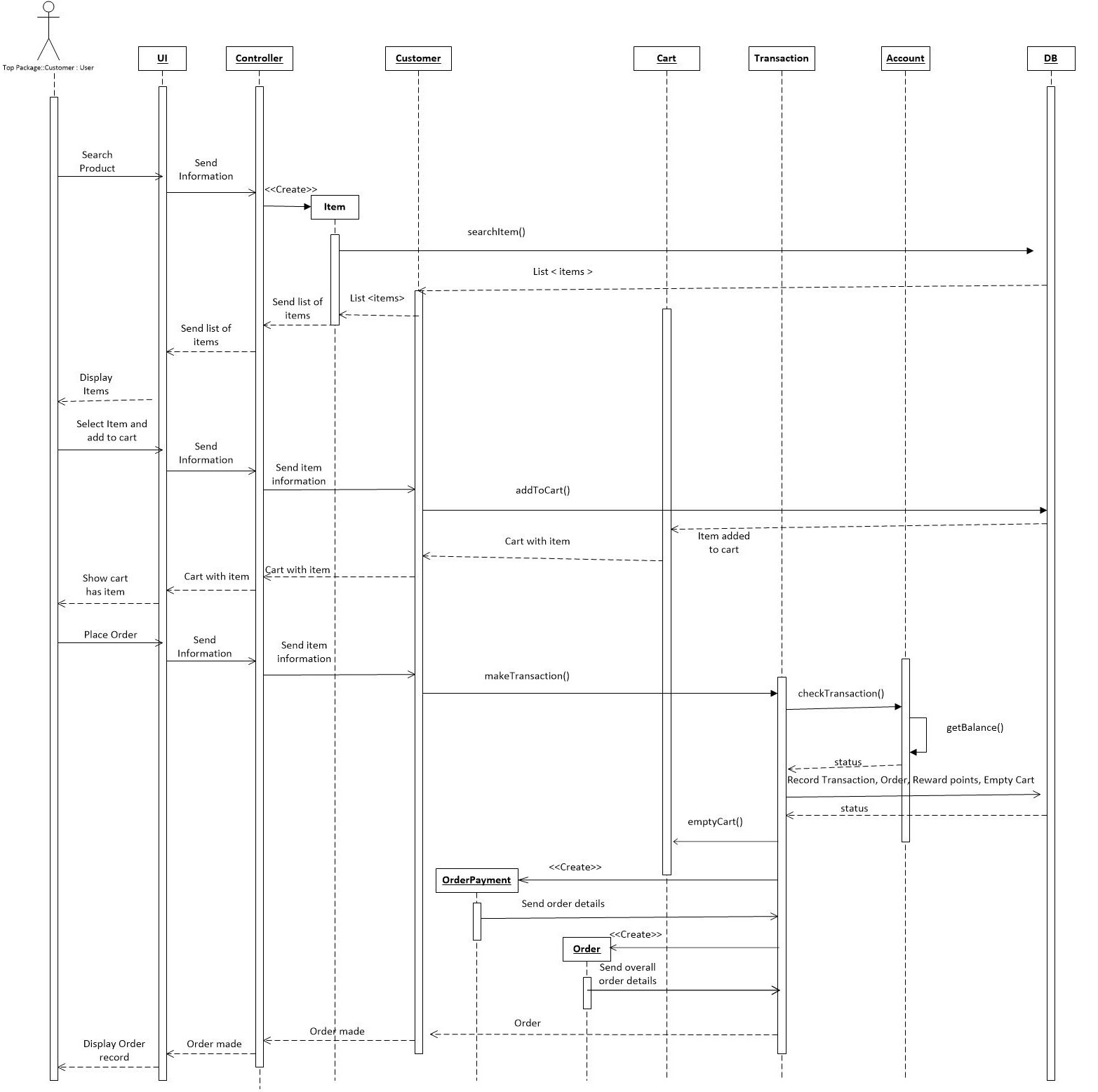
## Transfers

Customer has already logged in the system and he/she clicks on the transaction button on the user interface. Controller prompts customer to select the types of transfer a customer wants to make i.e. same account, Different account. When customer selects same account he/she will be prompted to select his/her own account to which transfer is to be made. If customer selects different account he/she will be prompted to enter the number of accounts and account details (from/to Account number and PIN) to which transfer is to be made. Customer object is created when he/she logs in the system. Controller then sends the information to customer object which initiates transaction. ‘MakeTransaction()’ is the method which handles the transaction.’CheckTransaction()’ method is called in ‘MakeTransaction()’ which queries database via customer account from which money is to be transferred, to check the balance. Is balance is insufficient system will send error message and UI will display to the customer an ‘invalid transaction message’. If the balance is greater than transaction amount then the amount is deducted from balance and same amount is added to account to which money is to be transferred. The list if transactions are passed to UI which displays the ‘transaction successful message’ to customer.



## Shopping Order

The customer is logged in to the system and chooses to shop for items. The customer first enters the desired item to be purchased and clicks on the search button of the UI. This will cause the creation of the Item object. The searchItem() is called through the Item object to search for the items in the Database. The list of Items is then retrieved and displayed to the customer. The customer then selects the items and clicks on ‘Add to Cart’. Controller sends the information to the Customer object, which calls the addToCart() method which interacts with the Database to update and retrieve the relevant entries with regard to the Items. The attributes of Cart object which is created for each Customer is updated and this information is sent back to the Customer object, and back to Controller and then forwarded to the UI which shows the customer of having items in his/her cart. The customer then clicks on ‘Proceed to checkout’ to place the order and make payment. The details are sent from UI to the Controller object which then initiates the Customer object to call makeTransaction() method . This method then calls the checkTransation() that checks if there is sufficient funds in the customer’s Account through another method called getBalance(). The status of the account is returned and then the OrderPayment and Order object is created to make updates. OrderPayment holds the details of the Customer’s accounts through which he/she chooses the payment and Order computes the total payment for an order from all the accounts to store the final order history for a customer. The order is thus made and the record is displayed to the customer.



# Tools & Programming Languages

1. JSP
2. Java (Application code)
3. Eclipse
4. PostgreSQL
5. JUnit (Java Unit Testing)
6. DBUnit ( Database Testing)
7. CodePro and Eclemma