|  |  |
| --- | --- |
| Checking/savings accounts  Project Design Document Revision 6 | Group 3  Lennon Brixey  Ken Machen  Conor Maginnis  Mathew Nielsen  UMUC CMSC 495 7981 Current Trends and Projects in Computer Science (2172) Professor Hung Dao |

|  |  |  |  |
| --- | --- | --- | --- |
| **REVISION #** | **DATE** | **DESCRIPTION** | **NAME** |
| **1** | **8 April 2017** | **Design Document** | **Lennon** |
| **2** | **8 April 2017** | **Class Diagram** | **Matthew** |
| **3** | **8 April 2017** | **Event-Trace Update** | **Ken** |
| **4** | **8 April 2017** | **Design Document Grammar** | **Ken** |
| **5** | **9 April 2017** | **Final Review** | **ALL** |
| **6** | **18 April 2017** | **Separated the Event Trace Diagrams and added scenarios with pre and post conditions** | **Ken** |

**CLASS DIAGRAM**

**Description:**

The class diagram shows our 3 classes as well as the methods and variables in each. The diagram also shows that the MainGUI class starts the SessionManager class. Also, the SessonManager class and Interest Calculator class have access to the information in the database.

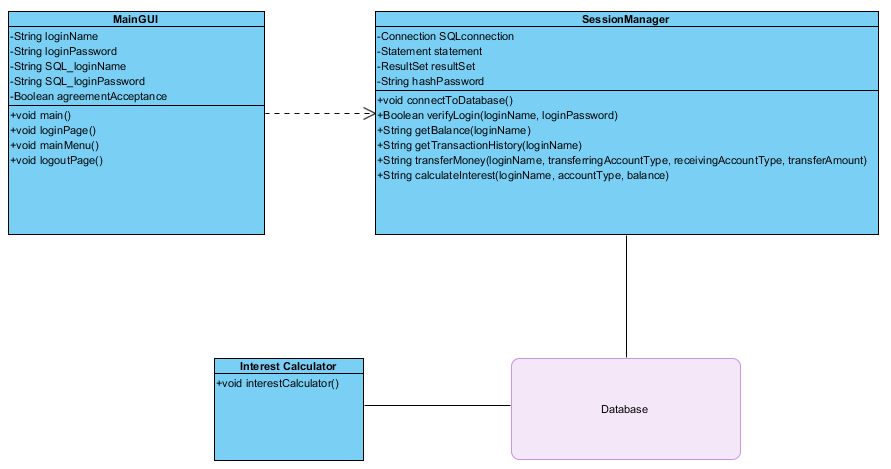
****

Figure 1 UML Class diagram

**EVENT-TRACE DIAGRAMS**

**Description:**

Figure 2 shows all the possible interactions from the user with the program and how those interactions affect the program. After initially opening the program the first set of instructions will be completed until the Main Menu GUI is shown. At this point, the other actions can be used.



Figure 2 Overall Event Trace Diagram

Scenario 1:

The user Opens the application and logs in.

Precondition: The computer is on and ready.

Post-condition: Successful login – user sees main menu. Unsuccessful login – user prompted to try again



Figure 3 Initial login sequence

Scenario 2:

User Transfers money from one account to another.

Pre-condition: User successfully logs in

Post-condition: If amount is not okay – user tries again. If amount is okay -user confirms transfer and sees the new balance.



Figure 4 Transfer sequence

Scenario 3:

The User checks account balance

Pre-condition: User successfully logged in.

Post-condition: User views balance.



Figure 5 Check Balance sequence

Scenario 4:

The User checks transaction history.

Pre-condition: Successful login

Post-condition: Views the transaction history



Figure 6 Transaction History sequence

Scenario 5:

User checks the interest earnings.

Pre-condition: Successful login.

Post-condition: User views the interest earnings



Figure 7 Show Interest sequence

Scenario 6:

The User logs out.

Pre-condition: Successful login.

Post-condition: Logged out



Figure 8 Logout sequence

**PSEUDOCODE**

**Description:**

The Pseudocode shows each class, method and variable name. It also shows each method call and a brief description of the events that need to occur inside those methods.

**Pseudocode:**

i) GUI subsystem

String loginName;

String loginPassword;

String SQL\_loginName;

String SQL\_loginPassword;

Boolean agreementAcceptance;

Void main{

connectToDatabase();

loginPage();

System.exit(0);

}

void loginPage(){

launch loginPage GUI

wait for input for the username and password

get username and password

Check if user name is valid using verifyLogin()

if yes, launch the user agreement

if no, display error message and keep waiting for input

launch acceptance agreement

when user agrees, launch mainMenu()

}

void mainMenu(){

launch mainMenu GUI

if user clicks “show balance”, display balance

if user clicks “transfer money”, transfer money to stated account

if user clicks “show transaction history”, transaction history is displayed

if user clicks “show interest earnings”. Interest earnings are showed

if user clicks “log out”, user is logged out and the logoutpage appears

}

void logoutPage(){

launch logoutPage

display log out message

start a timer for 10 seconds

after timer expires, close application

}

ii) Session Manager

Connection SQLconnection;

Statement statement;

ResultSet resultSet;

String hashPassword(loginPassword){

Hash password according to algorithm

}

void connectToDatabase(SQL\_loginName, SQL\_loginPassword){

set up connection with connectionURL, SQL\_loginName, and SQL\_loginPassword;

}

Boolean verifyLogin(loginName, loginPassword){

Create connection

Execute statement

If record is found, return true

If record is not found, return false

}

String getBalance(loginName){

Create connection

Execute statement

Return balance

}

Double getBalanceInterestCalculator(loginName){

Create connection

Execute statement

Return balance

}

String getTransactionHistory(loginName, accountType){

Create connection

Execute statement

Format string

Return transaction history

}

ResultSet getTransactionHistoryInterestCalculator(loginName, AccountType){

Create connection

Execute statement

Return transaction history

}

Boolean transferMoney(loginName, transferringAccountType, receivingAccountType, transferAmount){

Create connection

Get balance of both accounts

Account for the transfer

Execute statement updating database with new values

Return confirmation of transfer

}

String calculateInterest(loginName, accountType){

Get balance of account

Calculate interest based on account type and balance using Interest Calculator

Return the string with the information

}

Void updateBalance(loginName, accountType, amountToAdd){

Calculate the date

Create connection

Execute statement

}

iii) Interest Calculator

List<Transaction> savingHistory = **new** ArrayList<>();

List<Transaction> checkingHistory = **new** ArrayList<>();

**double** savingBalance = 0;

**double** checkingBalance = 0;

**double** savingRate = 0.06;

**double** checkingRate = 0.04;

**double** savingInterest, checkingInterest;

**int** daysinMonth;

Date generateDate = **new** Date();

**int** thisMonth = generateDate.~~getMonth~~()+1;

**int** thisDay = generateDate.~~getDate~~();

void interestCalculator(){

get all active accounts from database

get all balances for savings

loop through all savings accounts and calculate interest using calculateInterest()

get all balances for checking

loop through all checking accounts and calculate interest using calculateInterest()

}

private String removeChars(String date) {

remove the hyphen from the date

}

Double[] calculateInterest(loginName, accountType, balance){

get transaction history for last 30 days

find average balance of the last 30 days

calculate interest based on the average balance of the last 30 days

return interest amount

}

@Override

public String toString(){

format the string as a return;

return the string;

}

**UNRESOLVED RISKS**

**Description:**

Being aware of a potential risk in the design and generating possible mitigations before these problems arise will help eliminate these problems before they can interrupt production.

**Risks:**

1. Multiple logins of the same account to the database. Will the program allow multiple logins and how will the program ensure data integrity if two clients could possibly make changes to the account?

Mitigation: Add a login/logout variable and only allow one login to each account at a time.

1. Very small numbers in the interest calculator. It would be easy to have a calculation where the program has to deal in half cents (or smaller). Will these amounts be lost or saved in some other way?

Mitigation: Have a rounding system that is predictable and fair to the bank and customers.