**Homework project documentation**

Sooronbaev Adilet | APYFDL

***Description:***

* The project functionality is based on an ATM machine where a user can enter his bank account using his account number and pin code. After entering his credentials, another menu appears on the screen with various operations a user can choose, such as: *Check account balance*; *Cash withdrawal*; *Transfer money*; *Cash deposit*; *Show transactions*.
  + In case user chooses *Transfer money operation*, another menu appears where user must enter Account number of a person, he wants to transfer money to. After that, if user entered existing account number, another menu appears where he should enter amount of money he wants to transfer and currency in which he wants to transfer money. User can either confirm transfer or cancel it.
  + In case user chooses *Cash withdrawal operation*, new menu appears where user can choose amount of money in Forints he wants to withdraw from suggested amount, or he can choose to withdraw custom amount of money where user cam choose currency in which he wants to withdraw and amount of money.
  + In case user chooses *Cash deposit operation*, new menu appears where he can choose amount of money he wants to deposit and currency in which deposit is going to be processed.
  + In case user chooses *Check balance operation*, menu with his account number, name and balance appears.
  + In case user chooses *Show transactions* operation, if a user has any transactions, menu with all his transactions in table should appear, in other case user will be shown that he doesn’t have any transactions.
* The main currency in ATM is Hungarian Forint, but a user can also withdraw, deposit and transfer money in United States Dollar, where a conversion from HUF to USD will take place. A user will be notified of the current exchange rates in case of US Dollar withdrawal, deposit, or transfer.
* There are no fees for all the operations.

***\*Restrictions*:**

* Each bank account belongs to only one person, and a person can have at most one bank account with the same bank. All users' bank account numbers are unique.
* If a person tries to withdraw or transfer more money than he currently possesses in his bank account, the operation will fail with a message that he does not have sufficient money.
* The same will happen if a user tries to withdraw or transfer money in USD, but the converted amount from USD to HUF is greater than the amount of money a user has in his bank account. If a person enters incorrect account number or pin code, he will not enter his bank account in the ATM and the message that he entered wrong pin or account number appear.
* If during money transfer user enters non existing account to which he wants to transfer money, the message that user entered incorrect account number appears.

**Classes:**

* Bank

*Description:* Class which represents a bank and is responsible for storing accounts of the bank.

*Attributes:*

* *@Serial  
  private static final long* serialVersionUID = 6529685098267757690L;
  + I am specifying serialVersionUID in order for serialization to work properly. Because if I would not specify it everytime random serialVersionUID would be generated and thus causing errors with desearilization.
* *private final* String name; *//Bank name*
  + Name of the bank
* *private final* ArrayList<Account> accounts; *//Array of accounts in the bank*
  + ArrayList of accounts in the bank
* *private static* ArrayList<Integer> branch;
  + Represents a branch which contains all the account numbers, so there would be no similar account numbers in other banks

*Methods:*

* *private void* createAccount(Account account)
  + Method which adds new accounts to the ArrayList of accounts in Bank if such account does not exist
* *public* Account addHolder(String firstName, String lastName, *int* accountNumber, *double* balance, *int* pin)
  + Method which adds a new holder to the bank if holder with such account number doesn’t exist. Returns added account if it was successfully added
* *public* Account checkLogin(*int* accountNumber, *int* pin)
  + Method which checks if account with such account number and pin exists. Returns account with such account number and pin.
* *public* Account hasAccount(*int* accountNumber)
  + Checks if account with such account number exists in the bank. Returns account with such account number.
* *public* String getName()
  + Method which returns the name of the bank
* *public void* clearBranch()
  + Sets the branch ArrayList to be empty for testing.
* Account

*Description:* Represents account and stores account data, such as account number, pin, balance, bank, and user to which account belongs.

*Attributes:*

* *@Serial  
  private static final long* serialVersionUID = 6529685098267757690L;
  + Same is for this
* *private final int* accountNumber;
  + Account number of the account
* *private final* User holder;
  + User to whom account belongs
* *private double* balance;
  + Amount of money in the account
* *private final* ArrayList<Transactions> transactions;
  + List of transactions of the account
* *private final* Bank bank;
  + Bank to which account belongs
* *private final int* pin;
  + Pin of the account

Methods:

* *public int* getAccountNumber()
  + Returns number of the account
* *public double* getBalance()
  + Returns balance of the account
* *public void* addTransaction(*double* amount, Transactions transaction)
  + Adds new transaction to the ArrayList of transactions of this account
* *public boolean* transferMoney(*double* amount, Account receivingAccount, String currency)
  + Method which creates new transaction and calls method “*moneyTransfer(this, receivingAccount, amount ,newTransaction, currency)”* of class *Transaction* to transfer money from calling account to a receiving one with conversion to the specified currency. Returns true if transaction was successful.
* *public boolean* cashWithdrawal(*double* amount, String currency)
  + Method which creates new transaction and calls method “*cashWithdrawal(this, amount, newTransaction, currency)”* of class *Transaction* to withdraw money from calling account with conversion to the specified currency. Returns true if transaction was successful.
* *public boolean* cashDeposit(*double* amount, String currency)
  + Method which creates new transaction and calls method “*cashDeposit(this, amount, newTransaction, currency)”* of class *Transaction* to deposit money to the calling account with conversion to the specified currency. Returns true if transaction was successful.
* *public boolean* validate(*int* pin)
  + Returns true if pin matches to the pin of the account
* *public* Bank getBank()
  + Return bank to which account belongs to
* *public* ArrayList<Transactions> getTransactions()
  + Returns all transactions of the account
* *public* User getHolder()
  + Returns holder of the account
* Transactions

*Description:* Class that represents transactions of the user.

*Attributes:*

* *private final double* amount;
  + Amount of the transaction
* *private final* String name;
  + Name of the transactiwon
* Account account;
  + Account to which transaction belongs
* LocalDate currentDate;
  + Date of the transaction

*Methods:*

* *public boolean* moneyTransfer(Account transferringAccount, Account receivingAccount, *double* amount, Transactions newTransaction, String currency)
  + Method which adds *money transfer* transaction to the transferring account and *money receiving* transaction to the receiving account with the specified amount with conversion to specified currency. Returns true if transaction was successful.
* *public boolean* cashWithdrawal(Account account, *double* amount, Transactions newTransaction, String currency)
  + Method which adds *Cash withdrawal* transaction to the calling account in the amount specified with specified currency. Returns true if transaction was successful.
* *public boolean* cashDeposit(Account account, *double* amount, Transactions newTransaction, String currency)
  + Method which adds Cash deposit transaction to the calling account in the amount specified with specified currency. Returns true if transaction was successful.
* *public* String getName()
  + Returns name of the transaction.
* *public double* getAmount()
  + Returns amount of the transaction.
* *public* String getCurrentDate()
  + Return the date when transaction was made.
* User

Description: Class which represents user. Which has first and last name.

*Attributes:*

* *@Serial  
  private static final long* serialVersionUID = 6529685098267757690L;
  + Same for this
* *private final* String firstName;
  + First name of the user
* *private final* String lastName;
  + Last name of the user
* *private final* ArrayList<Account> accounts;
  + Accounts of the user

Methods:

* *public void* createAccount(Account newAccount)
  + Creates new account for the user
* *public* String getFirstName()
  + Returns first name of the user
* *public* String getLastName()
  + Returns last name of the user
* TransactionTable

*Description:* Class which stores transaction data in the table.

*Attributes:*

* *public* List<Object[]> transactions = *new* ArrayList<>();
  + List of user transactions
* *private final* String[] columns = {"Type", "Amount", "Date"};
  + Column names of the table

*Methods:*

* *public int* getRowCount()
  + Returns number of rows
* *public int* getColumnCount()
  + Returns number of columns
* *public* Object getValueAt(*int* rowIndex, *int* columnIndex)
  + Returns value in the specified row and column
* *public* String getColumnName(*int* column)
  + Returns the name of the column
* BankController

*Description:* Represents Controller of the Bank where all listeners for JButtons, JTextFields and other GUI elements are implemented.

*Method:*

* *public void* serializeBanks(ArrayList<Bank> banks)
  + Serializes banks after user manipulations in the program
* BankView

*Description:* Class which represents the GUI part of the program where all the windows and elements are initialized.

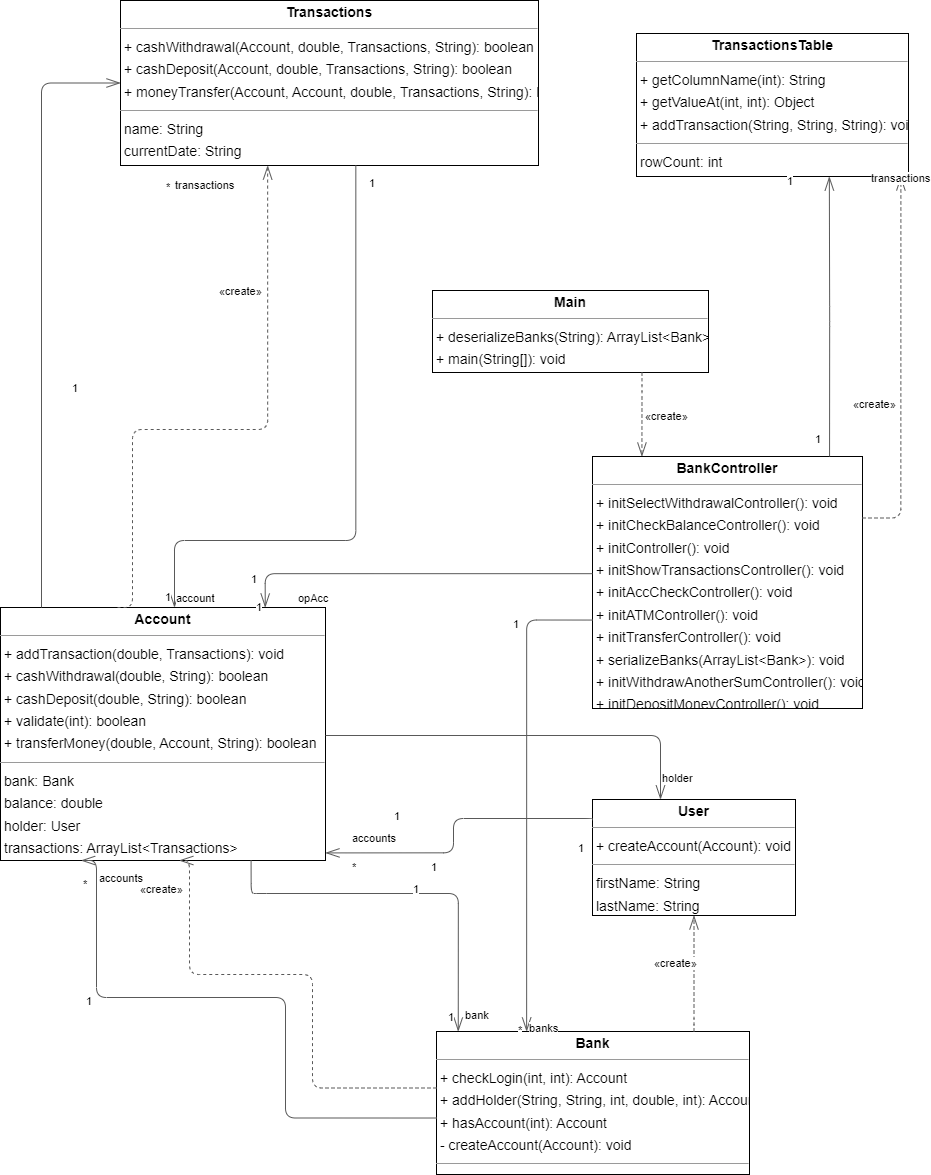
* Main

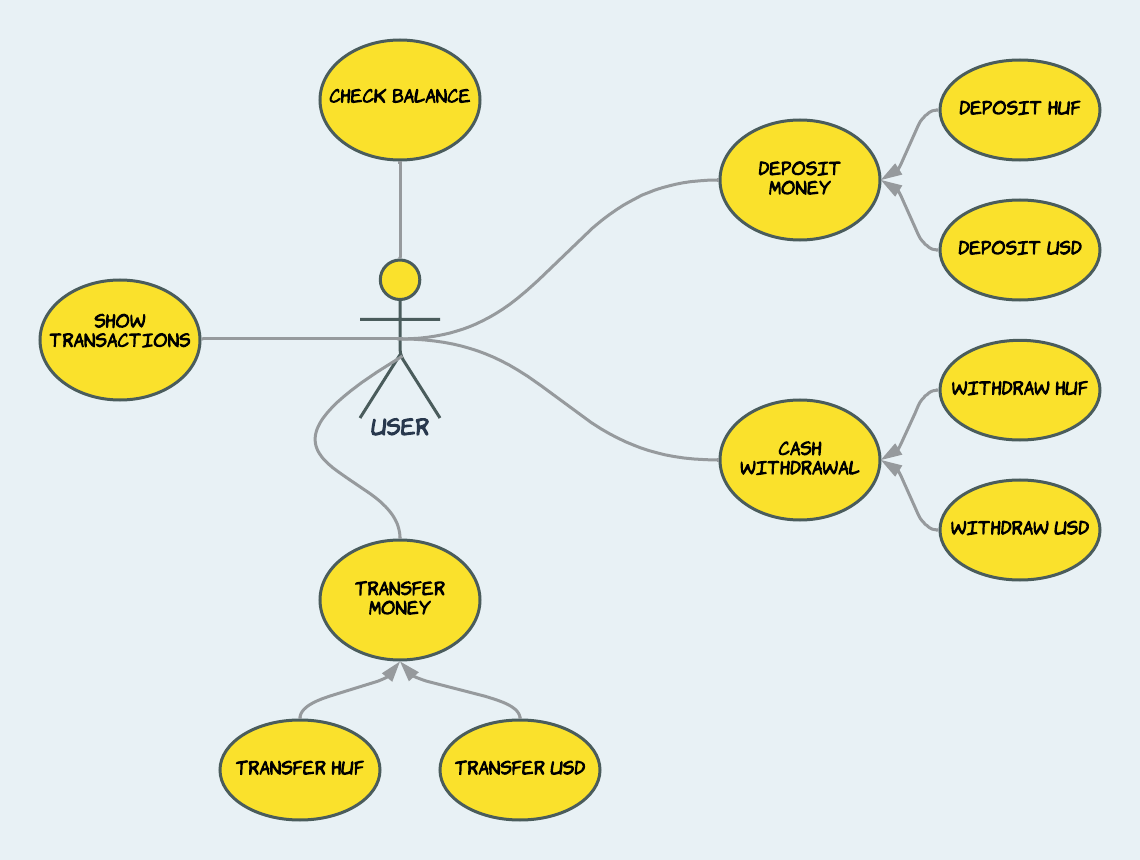
*Description:* Represent the main class

Methods:

* *public static void* main(String[] args)
  + The starting point for JVM to start execution of a Java program.
* *public static* ArrayList<Bank> deserializeBanks(String fileName)
  + Returns deserialized ArrayList of banks from file.

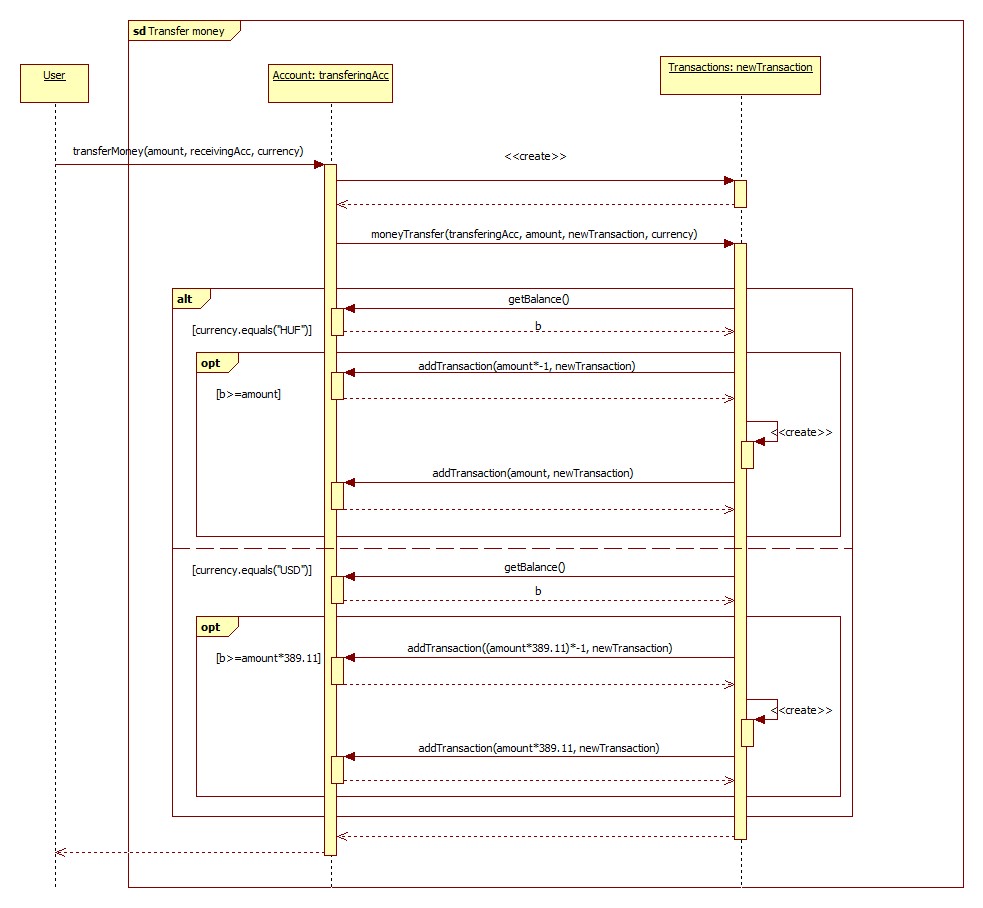
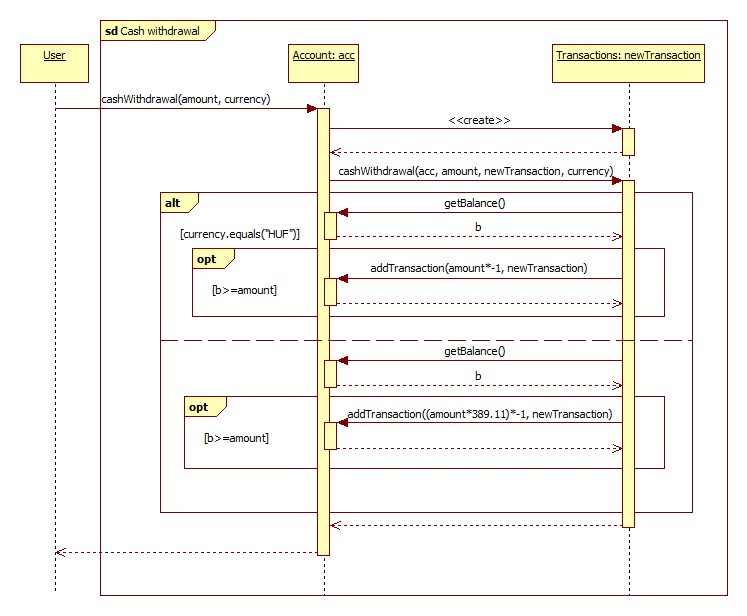
**Class diagram:**



**Use case diagram:**

* Actor: User
* Use cases: Transfer money, Show transactions, Deposit money, Check balance, Cash withdrawal

**Sequence diagrams:**

* Transfer money:
* Cash withdrawal:
* Diagram

  Description automatically generatedCash deposit:
* Check balance:

Diagram

Description automatically generated

* Diagram

  Description automatically generatedShow transactions:

Description of unit tests:

* BankTest
  + Tests:
    - *@Test  
      void* addSimilarHolder()
      * Should return null if holder with similar account number already exists
    - *@Test  
      void* addNewHolder()
      * Should return created account if account with such account number doesn't exist
    - *@Test  
      void* checkCorrectLogin()
      * Should return added account if account with such credentials exists
    - *@Test  
      void* checkIncorrectLogin()
      * Should return null if account with such credentials doesn't exist
    - *@Test  
      void* checkIncorrectPin()
      * Should return null if pin isn't correct
    - *@Test  
      void* checkIncorrectAccNum()
      * Should return null if account number isn't correct
    - *@Test  
      void* hasAccount()
      * Should return account, if such account number exists
    - *@Test  
      void* noHasAccount()
      * Should return null, if such account number doesn't exist
* AccountTest
  + Tests:
    - *@Test  
      void* transferLessHUF()
      * Should return true if the transferring amount in HUF is less than account balance
    - *@Test  
      void* transferMoreHUF()
      * Should return false if the transferring amount in HUF is greater than account balance
    - *@Test  
      void* transferLessUSD()
      * Should return true if the transferring amount in USD is less than account balance
    - *@Test  
      void* transferMoreUSD()
      * Should return false if the transferring amount in USD is greater than account balance
    - *@Test  
      void* cashWithdrawalLessHUF()
      * Should return true if amount of withdrawing in HUF is less than account balance
    - *@Test  
      void* cashWithdrawalMoreHUF()
      * Should return false if amount of withdrawing in HUF is greater than account balance
    - *@Test  
      void* cashWithdrawalLessUSD()
      * Should return true if amount of withdrawing in USD is less than account balance
    - *@Test  
      void* cashWithdrawalMoreUSD()
      * Should return false if amount of withdrawing in USD is greater than account balance
    - *@Test  
      void* validateCorrect()
      * Should return true if the entering pin is correct
    - *@Test  
      void* validateIncorrect()
      * should return false if the entering pin is incorrect
* MainTest
  + Tests
    - *@Test  
      void* deserializeBanks() *throws* IOException
      * Should return deserialized ArrayList of banks if the file with such name exists
    - *@Test  
      void* deserializeIncorrectBanks()
      * Should throw an exception if file with such name does not exist