Bank Assignment

Due: Monday June 4th at 5 pm

Console application:

Database:

* **USER\_STATUS**
  + USER\_STATUS\_ID (PK)
    - 1
    - 2
  + USER\_STATUS
    - ACTIVE
    - CLOSED
* **USER** table
  + USER\_ID (PK) 🡪 USE SEQUENCE TO GENERATE
  + USER\_FNAME
  + USER\_LNAME
  + USER\_PASWD
    - Add encrypt or hash feature.
  + USER\_DOB
  + USER\_TYPE\_ID (FK)
  + USER\_CREATED\_DATE
  + USER\_STATUS\_ID (FK)
* **USER\_TYPE** table – Lookup table - Only 2 types
  + USER\_TYPE\_ID (PK)
    - 1
    - 2
  + USER\_TYPE
    - “SUPERUSER”
    - “CUSTOMER”
* **BANKACCOUNT** table
  + BANK\_ACCOUNT\_ID (PK)
  + USER\_ID (FK)
    - A user can have multiple accounts.
  + BANK\_ACCOUNT\_BALANCE
    - This gets updated when user deposits, withdraws, transfer to and transfers from account.
  + BANK\_ACCOUNT\_CREATED\_DATE
  + BANK\_ACCOUNT\_STATUS\_ID (FK)
* **BANK\_ACCOUNT\_STATUS** table – Lookup table - Only 2 types
  + BANK\_ACCOUNT\_STATUS\_ID
    - 1
    - 2
  + BANK\_ACCONT\_STATUS
    - ACTIVE
    - CLOSED

BONUS:

* **TRANSACTIONS** table
  + TRANSACTION\_ID (PK) 🡪 USE SEQUENCE TO GENERATE
  + USER\_ID (FK)
    - A user can have many transactions.
  + TRANSACTION\_TYPE\_ID (FK)
  + TRANSACTION\_AMOUNT
  + TRANSACTION\_DATE
* **TRANSACTIONTYPE** table 🡪 Lookup table
  + TRANSACTION\_TYPE\_ID – ONLY 2 types
    - 1
    - 2
    - 3
  + TRANSACTION\_TYPE
    - “DEPOSIT”
    - “WITHDRAW”
    - “TRANSFER”
* **SEQUENCE:** Generate USER\_ID to increment
* **SEQUENCE:** Generate BANK\_ACCOUNT\_ID to increment
* **SEQUENCE:** Generated TRANSACTION\_ID to increment
* A registered user can login with their username and password
* An unregistered user can register by creating a username and password
* A superuser can view, create, update, and delete all users.
* A user can view their own existing accounts and balances.
* A user can create an account.
* A user can delete an account if it is empty.
* A user can add to or withdraw from an account.
* A user can execute multiple deposits or withdrawals in a session.
* A user can logout.
* **TOP\_MENU**
  + Type “EXIT” to exit program.
  + **REGISTER\_MENU**
    - Type “RETURN” to return to the top menu option.
    - Username:
    - Password:
    - First Name:
    - Last Name:
    - DOB:
      * Then check if username exists in database.
        + If username exists, then keep prompting for new username and checking. WRONG\_LOGIN\_EXCEPTION

Unless they type “RETURN”, then return to TOP\_MENU.

* + - * + If username does not exist:

Create new user().

User u = new User();

Create new User in database.

Send message that new user has been created and user can now log in.

Return to TOP Menu

* + **LOGIN MENU**
    - Type “RETURN” to back to TOP\_MENU.
    - Username?
    - Password?
      * **FIND USER\_ID** in USER table that has the same Username and Password.
        + **IF NOT FOUND**, print message, information is not found and ask for username and password again.
        + **IF FOUND**, using USERID: --**LOGIN\_FOUND\_MENU:**

**IF USER:**

View Accounts and Balances(USER\_ID)

Find User accounts by searching BANKACCOUNT TABLE for their BANK\_ACCOUNT\_ID using their USER\_ID.

Show only Accounts with an account status of 1 “Open”.

If account does not exist, message user that they do not have an account. NO\_ACCOUNT\_EXCEPTION

Return to LOGIN\_FOUND\_MENU:

Create An Account

Generate a new account with 0 balance.

Delete An Account If Empty

Show Accounts that are 0 balance and has account status of 1 “Open”.

Then prompt which Account# to delete.

If Database,

Deposit(depAmount, USER\_ID)

Withdraw(withAmount, USER\_ID)

Transfer(transAmount, USER\_ID)

Logout

**IF SUPERUSER:**

View Accounts And Balances of Users(USER\_ID)

Update Account(USER\_ID)

Create New User(USER\_ID)

Delete User(USER\_ID)

Logout()

* Use sequences to generate USER\_ID and BANK\_ACCOUNT\_ID.
* Throw custom exceptions in the event of user error (overdraft, incorrect password, etc).
* Provide validation messages through the console for all user actions.
* Use the DAO design pattern.
* Store superuser username/password and database connection information in a properties file.

Required technologies:

* PL/SQL with at least one stored procedure,
* JDBC with prepared and callable statements,
* Scanner for user input,
* JUnit tests on as much of the program as possible.

Bonus:

* A user's transactions are recorded.
* A user may view transaction history.

Required:

* Create a Maven project with your solution as JDBCBank, include it in your branch

* with your DB creation script (JDBCBank.sql).
* Use log4J to log data transactions and use JUnit for unit testing