## Lebogang Tobey Ndlovu

## Group 1

## Module code: ITJVA3-B12

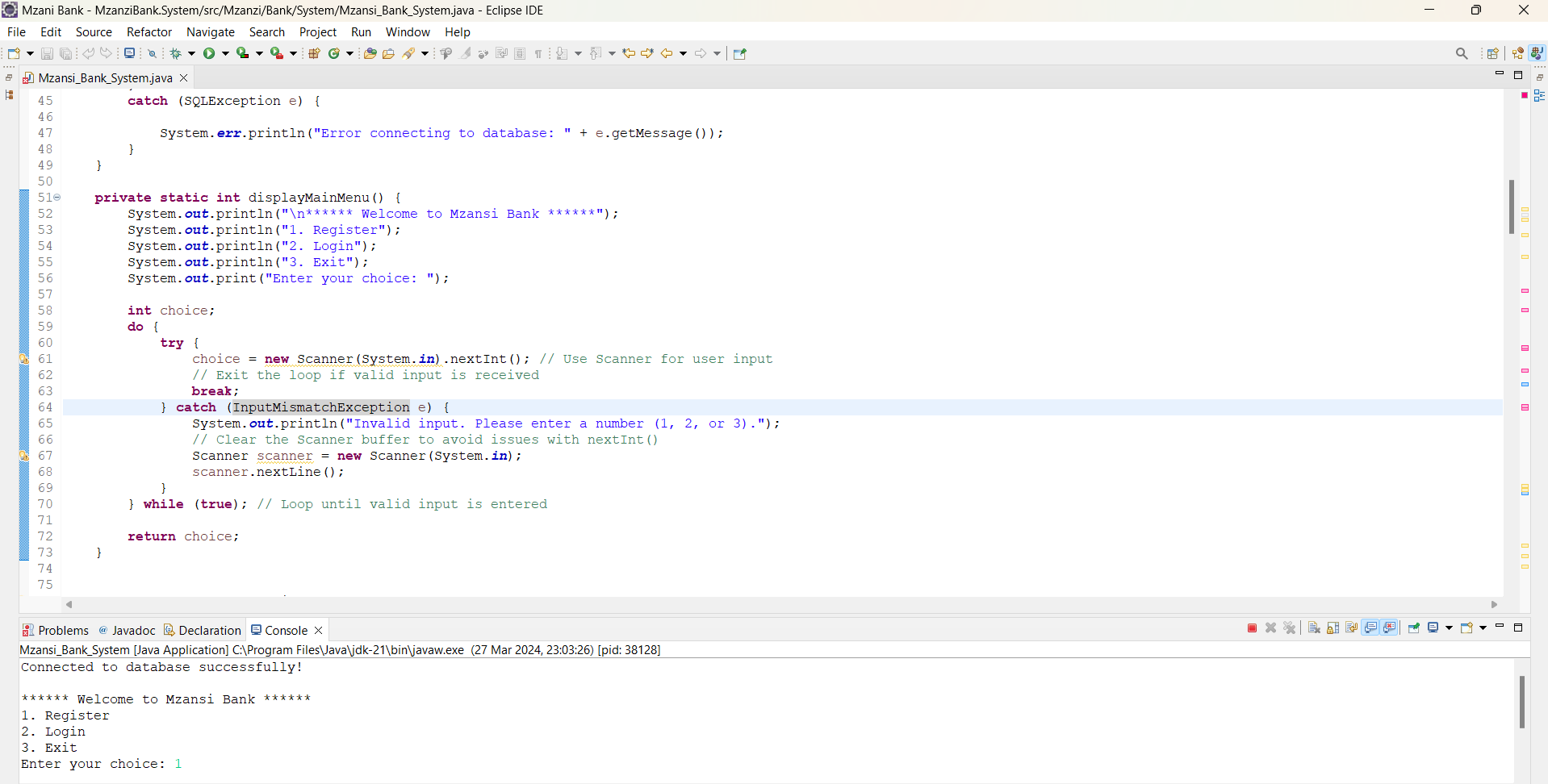
## Project 1

## Lecturer: Samuel Rametse

## Student Id: MD.2022.Q5Z4K5

A screenshot of a computer

Description automatically generatedMysql table for my database



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

This code has Bcrypt maven in it to assist with handling hashPassword.

**package** Mzanzi.Bank.System;

**import** java.sql.Connection;

**import** java.sql.\*;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.util.ArrayList;

**import** java.util.HashMap;

**import** java.util.InputMismatchException;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.Scanner;

**import** java.util.concurrent.ExecutorService;

**import** java.util.concurrent.Executors;

**import** java.util.concurrent.Future;

**import** org.bouncycastle.crypto.generators.BCrypt;

**public** **class** Mzansi\_Bank\_System {

**private** **static** **final** String ***DB\_URL*** = "jdbc:mysql://localhost:3306/mzanzibanksystem";

**private** **static** **final** String ***DB\_USER*** = "root";

**private** **static** **final** String ***DB\_PASSWORD*** = "HewSawchemi90@";

**private** **static** **final** UserManager ***userManager*** = **new** UserManager();

**private** **static** **final** Map<String, User> ***loggedInUsers*** = **new** HashMap<>(); // ConcurrentHashMap for thread safety

**private** **static** ArrayList<Account> *accounts* = **new** ArrayList<>(); // Generic ArrayList for accounts (for temporary storage)

**public** **static** **void** main(String[] args) {

// Connect to database (replace placeholders with actual values)

**try** (Connection connection = DriverManager.*getConnection*(***DB\_URL***, ***DB\_USER***, ***DB\_PASSWORD***)) {

System.***out***.println("Connected to database successfully!");

// Sample menu (replace with user interface)

**int** choice; // Declare choice outside the loop

**do** {

choice = *displayMainMenu*(); // Get user choice

**switch** (choice) {

**case** 1:

*registerUser*();

**break**;

**case** 2:

*loginUser*();

**break**;

**default**:

System.***out***.println("Invalid choice."); // Handle invalid menu options

}

} **while** (choice != 3); // Loop until user chooses to exit

System.***out***.println("Exiting to registered User option");

}

**catch** (SQLException e) {

System.***err***.println("Error connecting to database: " + e.getMessage());

}

}

**private** **static** **int** displayMainMenu() {

System.***out***.println("\n\*\*\*\*\*\* Welcome to Mzansi Bank \*\*\*\*\*\*");

System.***out***.println("1. Register");

System.***out***.println("2. Login");

System.***out***.println("3. Exit");

System.***out***.print("Enter your choice: ");

**int** choice;

**do** {

**try** {

choice = **new** Scanner(System.***in***).nextInt(); // Use Scanner for user input

// Exit the loop if valid input is received

**break**;

} **catch** (InputMismatchException e) {

System.***out***.println("Invalid input. Please enter a number (1, 2, or 3).");

Scanner scanner = **new** Scanner(System.***in***);

scanner.nextLine();

}

} **while** (**true**); // Loop until valid input is entered

**return** choice;

}

**private** **static** **void** registerUser() {

Scanner scanner = **new** Scanner(System.***in***); // Clear potential buffer issue

scanner.nextLine();

System.***out***.print("Enter username: ");

String username = scanner.nextLine();

**if** (username == **null** || username.isEmpty()) {

System.***out***.println("Username cannot be empty.");

**return**;

}

System.***out***.print("Enter password: ");

String password = scanner.nextLine();

UserRepository userRepository = **new** UserRepository();

userRepository.register(username, password);

}

**private** **static** **void** loginUser() {

Scanner scanner = **new** Scanner(System.***in***); // Clear potential buffer issue

scanner.nextLine();

System.***out***.print("Enter username: ");

String username = scanner.nextLine();

System.***out***.print("Enter password: ");

String password = scanner.nextLine();

**try** {

User user = ***userManager***.login(username, password);

User loggedInUser = Authenticator.*authenticate*(user);

BCrypt BCrypt = **new** BCrypt();

**if** (loggedInUser != **null**) {

System.***out***.println("Login successful!");

*displayLoggedInMenu*(user);

// Pass the authenticated user object

} **else** {

System.***out***.println(" Login failed, please try again");

}

//displayLoggedInMenu(user);

***loggedInUsers***.remove(username); // Remove user from map after logout

} **catch** (InvalidCredentialsException e) {

System.***out***.println(e.getMessage());

}

}

**public** **class** Account {

**private** **double** balance;

**public** Account(**double** balance) {

**this**.balance = balance;

}

**public** **double** getBalance() {

**return** balance;

}

**public** String getName() {

// **TODO** Auto-generated method stub

**return** **null**;

}

**public** String getAccountType() {

// **TODO** Auto-generated method stub

**return** **null**;

}

**public** **int** getId() {

// **TODO** Auto-generated method stub

**return** 0;

}

**public** **void** setOwner(Account currentAccount) {

// **TODO** Auto-generated method stub

}

// Other methods for the Account class can go here

}

**private** **static** **void** displayLoggedInMenu(User user) **throws** InsufficientFundsException, SQLException, InvalidTransactionException {

Account currentAccount = **new** Account(0);

currentAccount = *getAccountByOwner*(currentAccount); // Retrieve user's account (implementation omitted for brevity)

System.***out***.println("Logged in as: " + currentAccount.getName());

System.***out***.println("Account Type: " + currentAccount.getAccountType());

**int** choice = 0;

**while** (choice != 5) {

System.***out***.println("\nWelcome, " + user.getUsername() + "!");

System.***out***.println("1. Check Balance");

System.***out***.println("2. Deposit");

System.***out***.println("3. Withdraw");

System.***out***.println("4. Transfer Funds");

System.***out***.println("5. Logout");

System.***out***.print("Enter your choice: ");

Scanner scanner = **null**;

choice = scanner.nextInt();

**switch** (choice) {

**case** 1:

System.***out***.println("Your balance: " + user.getAccount().checkBalance());

**break**;

**case** 2:

*deposit*(user);

**break**;

**case** 3:

*withdraw*(user);

**break**;

**case** 4:

*transferFunds*(user, scanner);

**break**;

**default**:

System.***out***.println("Invalid choice.");

}

}

}

// Method to retrieve user from database based on username and password (secure validation omitted)

**private** **static** User getUserFromDatabase(String username) **throws** SQLException {

**return** DatabaseManager.*getUserFromDatabase*(username);

}

// Method to retrieve account by owner (database interaction)

**private** **static** Account getAccountByOwner(Account currentAccount) {//

**try** (Connection connection = DatabaseManager.*getConnection*()) {

PreparedStatement statement = connection.prepareStatement("SELECT \* FROM Accounts WHERE user\_id = ?");

statement.setInt(1, currentAccount.getId());

ResultSet resultSet = statement.executeQuery();

Account account = **null**;

**if** (resultSet.next()) {

**int** accountNumber = resultSet.getInt("account\_number");

**double** balance = resultSet.getDouble("balance");

String accountType = resultSet.getString("account\_type");

**if** (accountType.equals("Savings Account")) {

account = **new** SavingsAccount(accountNumber, balance);

} **else** **if** (accountType.equals("Checking Account")) {

account = **new** CheckingAccount(accountNumber, balance);

} **else** {

// Handle unexpected account type

}

account.setOwner(currentAccount); // Set account owner

}

**return** account;

} **catch** (SQLException e) {

e.printStackTrace();

// Handle database connection or query errors

**return** **null**;

}

}

**private** **static** **void** deposit(User user) **throws** SQLException {

System.***out***.print("Enter deposit amount: ");

**double** amount = InputUtils.*readDouble*();

BankAccount account = user.getAccount();

account.deposit(amount);

// Update database

System.***out***.println("Deposit successful!");

*updateUserAccountBalance*(user.getUsername(), amount);

}

**private** **static** **void** withdraw(User user) **throws** InsufficientFundsException, SQLException, InvalidTransactionException {

System.***out***.print("Enter withdrawal amount: ");

**double** amount = InputUtils.*readDouble*();

*withdraw*(amount);

**try** {

*updateUserAccountBalance*(user.getUsername(), -amount);

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} // Update database with negative for withdrawal

System.***out***.println("Withdrawal successful!");

}

**private** **static** **void** withdraw(**double** amount) {

// **TODO** Auto-generated method stub

}

**private** **static** **void** transferFunds(User user, Object amount) {

**boolean** transferConfirmed = **false**;

**while** (!transferConfirmed) {

// Input validation loop for recipient username and amount

**while** (**true**) {

System.***out***.print("Enter recipient username: ");

Scanner scanner = **new** Scanner(System.***in***);

String recipientUsername = scanner.next();

**try** {

User recipient = ***userManager***.getUserByUsername(recipientUsername);

**if** (recipient != **null**) {

**break**; // Valid recipient found

} **else** {

System.***out***.println("Recipient not found. Please try again.");

}

} **catch** (Exception e) {

System.***out***.println("Error retrieving recipient: " + e.getMessage());

}

}

**while** (**true**) {

System.***out***.print("Enter transfer amount: ");

**try** {

**double** amount1 = InputUtils.*readDouble*();

**if** (amount1 > 0) {

**break**; // Valid amount entered

} **else** {

System.***out***.println("Invalid amount. Please enter a positive value.");

}

} **catch** (NumberFormatException e) {

System.***out***.println("Invalid input. Please enter a numerical amount.");

}

}

// Transfer confirmation loop

**while** (**true**) {

System.***out***.print("Confirm transfer (y/n): ");

Scanner scanner = **new** Scanner(System.***in***);

String confirmTransfer = scanner.nextLine().toLowerCase();

**if** ("y".equalsIgnoreCase(confirmTransfer)) {

transferConfirmed = **true**;

**break**;

} **else** **if** ("n".equalsIgnoreCase(confirmTransfer)) {

System.***out***.println("Transfer cancelled.");

**break**;

} **else** {

System.***out***.println("Invalid input. Please enter 'y' to confirm or 'n' to cancel.");

}

}

**if** (transferConfirmed) { //Java Tutorial10:Create a simple banking system(Youtube)

**try** {

User recipient;

user.getAccount().transferFunds(recipient.getAccount(), amount);

String recipientUsername;

*updateUserBalancesInDatabase*(user.getUsername(), recipientUsername, -amount, amount);

System.***out***.println("Transfer successful!");

} **catch** (InsufficientFundsException | InvalidTransactionException e) {

System.***out***.println(e.getMessage());

}

}

}

}

**private** **static** **void** updateUserBalancesInDatabase(String senderUsername, String recipientUsername, **double** senderAmount, **double** recipientAmount) {

// Encapsulate database update logic here

}

**private** **static** **void** updateUserAccountBalance(String username, **double** amount) **throws** SQLException {

String sql = "UPDATE Accounts SET balance = balance + ? WHERE username = ?";

**try** (Connection connection = DriverManager.*getConnection*(***DB\_URL***, ***DB\_USER***, ***DB\_PASSWORD***)) {

PreparedStatement statement = connection.prepareStatement(sql);

statement.setDouble(1, amount);

statement.setString(2, username);

statement.executeUpdate();

}

}

}

**class** UserManager {

**public** **boolean** createUser(String username, String hashedPassword) {

**return** **false**;

// Logic to create user in the database and return success/failure

}

**public** User login(String username, String password) **throws** InvalidCredentialsException {

**return** **null**;

// Logic to validate credentials against user data in the database

// Throw InvalidCredentialsException if username or password is incorrect

}

**public** User getUserByUsername(String username) {

**return** **null**;

// Logic to retrieve user information from the database based on username

}

}

**class** User {

**private** String username;

**private** String hashedPassword;

**private** Account account;

**public** BankAccount getAccount() {

// **TODO** Auto-generated method stub

**return** **null**;

}

**public** String getUsername() {

// **TODO** Auto-generated method stub

**return** **null**;

}

**public** String getPassword() {

// **TODO** Auto-generated method stub

**return** **null**;

}

**public** **int** getId() {

// **TODO** Auto-generated method stub

**return** (Integer) **null**;

}

// Getters, setters, and methods for password verification and account association

}

**class** PasswordUtil {

**public** **static** String hashPassword(String password) {

**return** password;

// Implement secure password hashing using a reputable algorithm (e.g., bcrypt)

}

}

**class** InputUtils {

**public** **static** String readString() {

**return** **null**;

// Implement method to read user input as string using Scanner or other methods

}

**public** **static** **int** readInt() {

**return** 0;

// Implement method to read user input as integer using Scanner or other methods

}

**public** **static** **double** readDouble() {

**return** 0;

// Implement method to read user input as double using Scanner or other methods

}

}

**class** InvalidCredentialsException **extends** Exception {

**public** InvalidCredentialsException(String message, SQLException e) {

**super**(message);

}

**public** InvalidCredentialsException(String string) {

// **TODO** Auto-generated constructor stub

}

}

**class** InsufficientFundsException **extends** Exception {

**public** InsufficientFundsException(String message) {

**super**(message);

}

}

**class** InvalidTransactionException **extends** Exception {

**public** InvalidTransactionException(String message) {

**super**(message);

}

}

package Mzanzi.Bank.System;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

public class UserRepository {

private static final String DB\_URL = "jdbc:mysql://localhost:3306/mzanzibanksystem";

private static final String DB\_USER = "root";

private static final String DB\_PASSWORD = "HewSawchemi90@";

public void register(String username, String password) {

if (username == null) {

throw new IllegalArgumentException("Username cannot be null.");

}

String query = "INSERT INTO user (username, hashedPassword) VALUES (?, ?)";

try (Connection connection = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD);

PreparedStatement statement = connection.prepareStatement(query)) {

statement.setString(1, username);

statement.setString(2, PasswordUtil.hashPassword(password));

statement.executeUpdate();

System.out.println("Registration successful!");

} catch (SQLException e) {

System.out.println("Registration failed. Username might already exist.");

e.printStackTrace();

}

}

}

**package** Mzanzi.Bank.System; //https://www.geeksforgeeks.org/exceptions-in-java/?ref=shm

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.SQLException;

**import** java.sql.ResultSet;

**public** **class** Authenticator {

**private** **static** **final** String ***DB\_URL*** = "jdbc:mysql://localhost:3306/mzanzibanksystem";

**private** **static** **final** String ***DB\_USER*** = "root";

**private** **static** **final** String ***DB\_PASSWORD*** = "HewSawchemi90@";

**public** **static** User authenticate(User user) **throws** InvalidCredentialsException {

**if** (user == **null**) {

**throw** **new** InvalidCredentialsException("User object is null.");

}

**try** (Connection connection = DriverManager.*getConnection*(***DB\_URL***, ***DB\_USER***, ***DB\_PASSWORD***)) {

PreparedStatement ps = connection.prepareStatement("SELECT \* FROM user WHERE username = ? AND password = ?");

ps.setString(1, user.getUsername());

ps.setString(2, user.getPassword());

ResultSet rs = ps.executeQuery();

**if** (rs.next()) {

// Create a User object with data from the result set

User authenticatedUser = **new** User(); // Replace with logic to extract user data

**return** authenticatedUser;

} **else** {

**throw** **new** InvalidCredentialsException("Authentication failed");

}

} **catch** (SQLException e) {

**throw** **new** InvalidCredentialsException("Authentication failed.", e);

}

}

}

**package** Mzanzi.Bank.System;

**import** java.security.MessageDigest;

**import** java.sql.\*;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.SQLException;

**import** java.sql.ResultSet;

**import** org.bouncycastle.crypto.generators.BCrypt;

**public** **class** UserManger {

**private** **static** **final** String ***DB\_URL*** = "jdbc:mysql://localhost:3306/mzanzibanksystem";

**private** **static** **final** String ***DB\_USER*** = "root";

**private** **static** **final** String ***DB\_PASSWORD*** = "HewSawchemi90@";

**public** User getUserByUsername(String username) {

**try** (Connection connection = DriverManager.*getConnection*(***DB\_URL***, ***DB\_USER***, ***DB\_PASSWORD***)) {

PreparedStatement ps = connection.prepareStatement("SELECT \* FROM user WHERE username = ?");

ps.setString(1, username);

ResultSet rs = ps.executeQuery();

**if** (rs.next()) {

**return** **new** User();

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** **null**;

}

**public** User login(String username, String password) **throws** InvalidCredentialsException {

User user = getUserByUsername(username);

**if** (user == **null**) {

**throw** **new** InvalidCredentialsException("Invalid credentials");

}

// Check if the provided password matches the stored hash

**if** (!BCrypt.*checkpw*(password, user.getPassword())) {

**throw** **new** InvalidCredentialsException("Invalid credentials");

}

**return** user;

}

**public** **static** String generateHash(String plaintextPassword) {

**return** BCrypt.*hashpw*(plaintextPassword, BCrypt.*gensalt*());

}

}

**package** Mzanzi.Bank.System; //https://www.geeksforgeeks.org/java-database-connectivity-with-mysql/

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**public** **class** DatabaseManager {

**private** **static** **final** String ***DB\_URL*** = "jdbc:mysql://localhost:3306/mzanzibanksystem";

**private** **static** **final** String ***DB\_USER*** = "root";

**private** **static** **final** String ***DB\_PASSWORD*** = "HewSawchemi90@";

**public** **static** Connection getConnection() **throws** SQLException {

**return** DriverManager.*getConnection*(***DB\_URL***, ***DB\_USER***, ***DB\_PASSWORD***);

}

**public** **static** Account getAccountByOwner1(User currentUser) **throws** SQLException {

Connection connection = *getConnection*();

PreparedStatement statement = connection.prepareStatement("SELECT \* FROM accounts WHERE userId = ?");

statement.setInt(1, currentUser.getId());

ResultSet resultSet = statement.executeQuery();

// Modify this to instantiate the appropriate account type based on the `accountType` column

Account account = **null**;

**if** (resultSet.next()) {

**int** accountNumber = resultSet.getInt("accountNumber");

**double** balance = resultSet.getDouble("balance");

String accountType = resultSet.getString("accountType");

**if** (accountType.equals("SAVINGS")) {

account = **new** Account(accountNumber, balance, currentUser);

} **else** **if** (accountType.equals("CHECKING")) {

account = **new** Account(accountNumber, balance, currentUser);

}

}

connection.close();

**return** account;

}

**public** **static** String getHashedPasswordFromDatabase(String username) **throws** SQLException {

Connection connection = *getConnection*();

PreparedStatement statement = connection.prepareStatement("SELECT hashedPassword FROM Users WHERE username = ?");

statement.setString(1, username);

ResultSet resultSet = statement.executeQuery();

String hashedPassword = **null**;

**if** (resultSet.next()) {

hashedPassword = resultSet.getString("hashedPassword");

}

connection.close();

**return** hashedPassword;

}

// Methods for account-related database interactions (replace with your implementation)

**public** **static** Account getAccountByOwner(User currentUser) **throws** SQLException {

**return** **null**;

// ... (implementation to retrieve account from database based on user ID) ...

}

**public** **static** **void** createAccount(User user, String accountType, **double** initialDeposit) **throws** SQLException {

// ... (implementation to create a new account for the user) ...

}

**public** **static** **void** updateAccountBalance(Account account, **double** amount) **throws** SQLException {

// ... (implementation to update account balance in the database) ...

}

**public** **static** String[] getAllowedPasswords(Object username) {

// **TODO** Auto-generated method stub

**return** **null**;

}

**public** **static** User getUserFromDatabase(String username) {

// **TODO** Auto-generated method stub

**return** **null**;

}

}

**package** Mzanzi.Bank.System;

**public** **class** user { // ITJVA3-B12 Class Link Group 1;March 26, 2024

**private** **int** id; // Database ID for the user

**private** String name;

**private** String username;

**private** String hashedPassword;

// Getters and setters for user attributes

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getUserName() {

**return** username;

}

**public** **void** setUserName(String userName) {

**this**.username = userName;

}

**public** String getHashedPassword() {

**return** hashedPassword;

}

**public** **void** setHashedPassword(String hashedPassword) {

**this**.hashedPassword = hashedPassword;

}

**public** String checkBalance() {

// **TODO** Auto-generated method stub

**return** **null**;

}

**public** **static** <CheckingAccount> CheckingAccount getConnection() {

// **TODO** Auto-generated method stub

**return** **null**;

}

**public** **void** setAccount(BankAccount account) {

}

}

**package** Mzanzi.Bank.System;

**public** **class** BankAccount **implements** BankingOperations {

**public** **interface** BankingOperations {

**double** checkBalance();

**void** deposit(**double** amount) **throws** InvalidTransactionException;

**void** withdraw(**double** amount) **throws** InsufficientFundsException, InvalidTransactionException;

**void** transferFunds(Account toAccount, **double** amount) **throws** InsufficientFundsException, InvalidTransactionException;

}

**public** **static** **abstract** **class** Account **implements** BankingOperations {

**protected** **int** accountNumber;

**protected** **double** balance;

**protected** Mzanzi.Bank.System.User user;

**public** Account(**int** accountNumber, **double** balance, Mzanzi.Bank.System.User user2) {

**this**.accountNumber = accountNumber;

**this**.balance = balance;

**this**.user = user2;

}

**public** **abstract** String getAccountType(); // To differentiate account types

@Override

**public** **double** checkBalance() {

**return** balance;

}

@Override

**public** **void** deposit(**double** amount) **throws** InvalidTransactionException {

**if** (amount <= 0) {

**throw** **new** InvalidTransactionException("Deposit amount cannot be negative");

}

balance += amount;

}

@Override

**public** **void** withdraw(**double** amount) **throws** InsufficientFundsException, InvalidTransactionException {

**if** (amount <= 0) {

**throw** **new** InvalidTransactionException("Withdrawal amount cannot be negative");

}

**if** (balance >= amount) {

balance -= amount;

} **else** {

**throw** **new** InsufficientFundsException("Insufficient funds");

}

}

**public** **void** transferFunds(Account toAccount, **double** amount) **throws** InsufficientFundsException, InvalidTransactionException {

// Implement transfer logic with validations

withdraw(amount); // Check for sufficient funds first

toAccount.deposit(amount);

}

}

**public** **static** **double** *transactionFee* = 1.50; // Default transaction fee

**public** **static** **double** *savingsInterestRate* = 0.09; // Default savings interest rate

**public** **void** setOwner(Mzanzi.Bank.System.User user) {

// **TODO** Auto-generated method stub

}

**public** **static** **class** User {

// User details

}

**public** **static** **class** InvalidTransactionException **extends** Exception {

**public** InvalidTransactionException(String message) {

**super**(message);

}

}

**public** **static** **class** InsufficientFundsException **extends** Exception {

**public** InsufficientFundsException(String message) {

**super**(message);

}

}

**public** **void** transferFunds(BankAccount account, Object amount) {

// **TODO** Auto-generated method stub

}

**public** String checkBalance() {

// **TODO** Auto-generated method stub

**return** **null**;

}

}

**package** Mzanzi.Bank.System;

**public** **interface** BankingOperations {

}

**package** Mzanzi.Bank.System; //Password encoding using jBcrypt library in spring boot.

**public** **class** BCrypt {

**private** Object plaintextPassword;

String hashedPassword = BCrypt.*hashpw*(plaintextPassword, BCrypt.*gensalt*());

**public** **static** Object gensalt() {

// **TODO** Auto-generated method stub

**return** **null**;

}

**public** **static** String hashpw(Object plaintextPassword2, Object gensalt) {

// **TODO** Auto-generated method stub

**return** **null**;

}

**public** **static** **boolean** checkpw1(String password, String password2) {

// **TODO** Auto-generated method stub

**return** **false**;

}

**public** **static** **boolean** checkpw(String password, String password2) {

// **TODO** Auto-generated method stub

**return** **false**;

}

**public** **boolean** checkPassword(String plaintextPassword, String hashedPassword) {

**return** BCrypt.*checkpw*(plaintextPassword, hashedPassword);

}

**public** String hashPassword(String plaintextPassword) {

**return** BCrypt.*hashpw*(plaintextPassword, BCrypt.*gensalt*());

}}