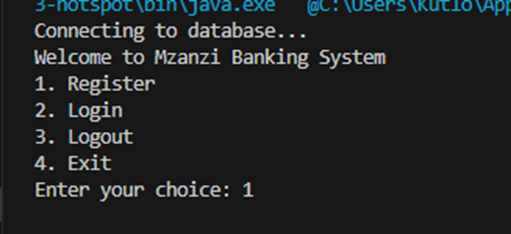
**ITJVA3-12 project.**

**Kutloano Molefe MD.2022.Q8Q4M2**

**Pictures of output**

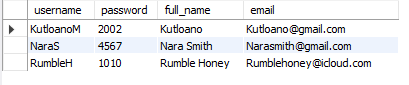
**Sample menue :Before login and/ or registration**

****

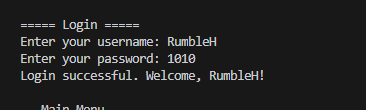
**After registering the user. They are now stored in the database user table**

**Registering a user**

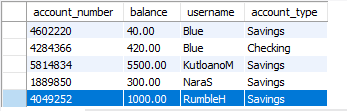
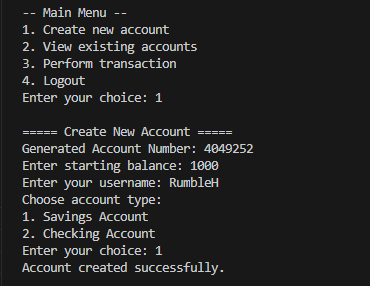
**A screen shot of a computer

Description automatically generated**

**Now logging in with the registered user**

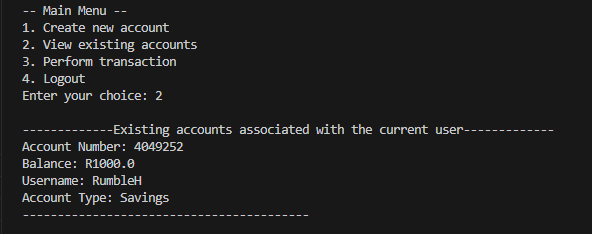
****

**Sample menue:after logging in and creating a new account**

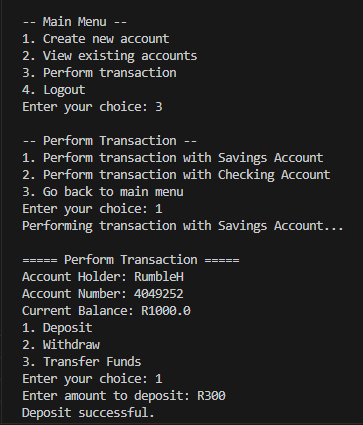
****

**After a new account is created it is stored in the database in the accounts table**

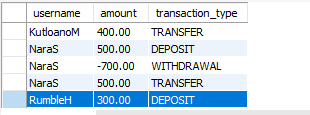
**Option 2:View existing Accounts for user that is currently logged in**

****

**Option 3: Performing Transactions(we are depositing money in this transaction)**

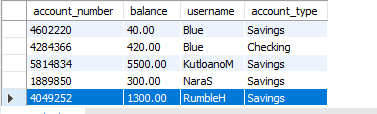
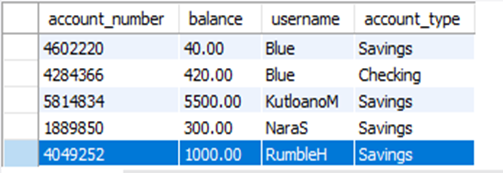
****

**In the transaction table in our database the deposit we made is shown**

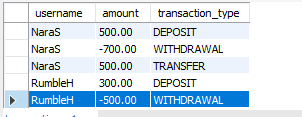
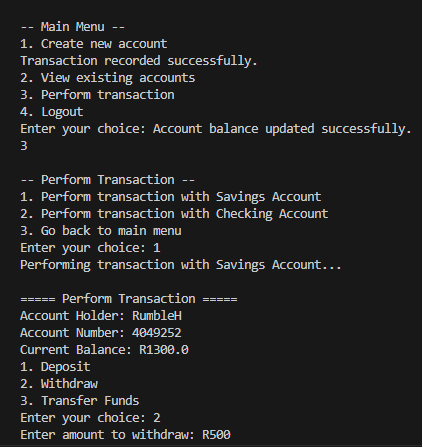
****

**In our database because we deposited 500 it is now updated to a new balance of 1500 in our accounts table**

**Before After**

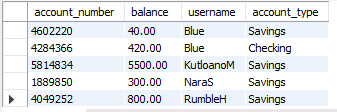
****

**In this transaction we are withdrawing**

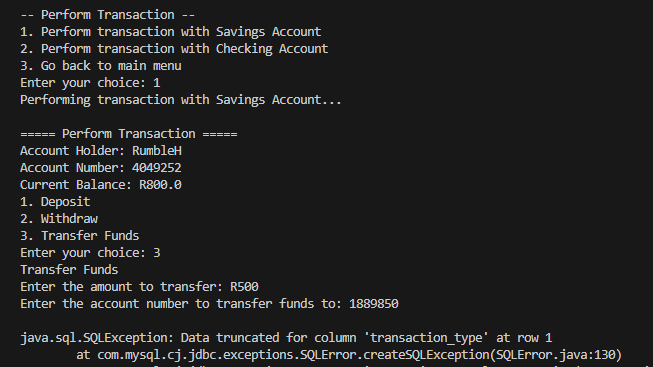
****

Transaction table in our database shows our withdrawal

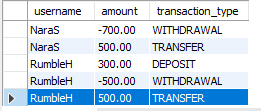
**New balance is updated in the database**

****

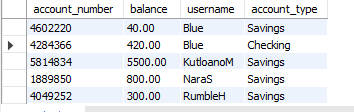
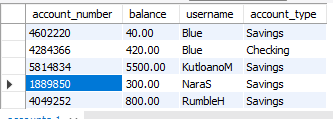
**In this transaction we are transferring funds into an account number entered by the user (shows an error in this section but still updates in our database accordingly)**

****

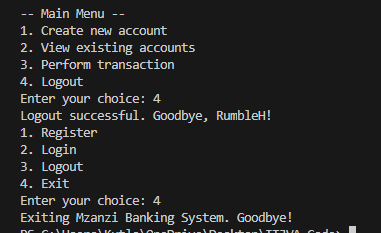
**Transaction table shows our transaction transfer**

****

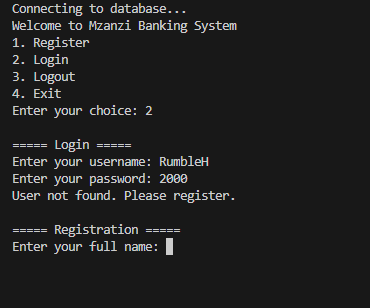
**Database updates the user we transferred to amount:since we transferred to KutloanoM the previously had 5000 now they have 5500**

****

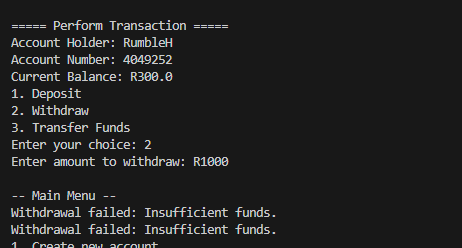
**Log out and exist options**

****

**Authentication:displaying if you get the password incorrect you cant login**

****

**Withdrawal if there is insufficient funds**

****

**Code:**

//MzansiBankingSystem.java

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.HashMap;

import java.util.Map;

import java.util.Random;

import java.util.Scanner;

public class MzanziBankingSystem {

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

    static Map<String, String> userCredentials = new HashMap<>();

    static String currentUser = null;

    static Connection connection = null;

    static final String DB\_URL = "jdbc:mysql://localhost:3306/newdb";//reference: https://www.javatpoint.com/example-to-connect-to-the-mysql-database

    static final String USER = "root";

    static final String PASS = "Kutli@2002";

    public static void main(String[] args) {

        try {

            Class.forName("com.mysql.cj.jdbc.Driver");

            System.out.println("Connecting to database...");

            connection  = DriverManager.getConnection(DB\_URL, USER, PASS);

        }

        catch (Exception e) {

            System.out.print("Error;"+e);

            return;

        }

        System.out.println("Welcome to Mzanzi Banking System");

        boolean exit = false;

        while (!exit) {

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

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

            System.out.println("3. Logout");

            System.out.println("4. Exit");

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

            int choice = 0;

            try {

                choice = scanner.nextInt();

            } catch (Exception e) {

                System.out.println("Invalid input. Please enter a number.");

                scanner.next();

                continue;

            }

            scanner.nextLine();

            switch (choice) {

                case 1:

                    register();

                    break;

                case 2:

                    login();

                    break;

                case 3:

                    logout();

                    break;

                case 4:

                    exit = true;

                    System.out.println("Exiting Mzanzi Banking System. Goodbye!");

                    break;

                default:

                    System.out.println("Invalid choice. Please enter a number between 1 and 4.");

            }

        }

    }

//reference: https://www.javaguides.net/2020/03/java-scanner-tutorial-reading-login-and-registration-user-input.html

    public static void register() {

        System.out.println("\n===== Registration =====");

        System.out.print("Enter your full name: ");

        String fullName = scanner.nextLine();

        System.out.print("Enter your email address: ");

        String email = scanner.nextLine();

        System.out.print("Enter your desired username: ");

        String username = scanner.nextLine();

        System.out.print("Enter your desired password: ");

        String password = scanner.nextLine();

        try {

            String query = "INSERT INTO users (username, password,full\_name,email) VALUES (?, ?, ?, ?)";//reference: https://stackoverflow.com/questions/3555275/using-mysqls-in-statement-in-java

            PreparedStatement preparedStatement = connection.prepareStatement(query);

            preparedStatement.setString(1, username);

            preparedStatement.setString(2, password);

            preparedStatement.setString(3,fullName);

            preparedStatement.setString(4, email);

            int rowsInserted = preparedStatement.executeUpdate();

            if (rowsInserted > 0) {

                System.out.println("Registration successful for user: " + username);

            } else {

                System.out.println("Registration failed. Please try again.");

            }

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

    public static void login() {

        System.out.println("\n===== Login =====");

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

    String username = scanner.nextLine();

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

    String password = scanner.nextLine();

    try {

        String query = "SELECT \* FROM users WHERE username = ? AND password = ?";//reference:https://stackoverflow.com/questions/7598082/how-to-set-placeholders-in-like-statement-query

        PreparedStatement preparedStatement = connection.prepareStatement(query);

        preparedStatement.setString(1, username);

        preparedStatement.setString(2, password);

        ResultSet resultSet = preparedStatement.executeQuery();

        if (resultSet.next()) {

            String sql = "SELECT \* FROM users WHERE username = ? AND password = ?";

            PreparedStatement preparedStatemen = connection.prepareStatement(sql);

            preparedStatemen.setString(1, username);

            preparedStatemen.setString(2, password);

            ResultSet resultSett = preparedStatemen.executeQuery();

            if (resultSett.next()) {

                currentUser = username;

                System.out.println("Login successful. Welcome, " + username + "!");

                displayMainMenu();

            } else {

                System.out.println("Login failed. Please check your credentials.");

            }

        } else {

            System.out.println("User not found. Please register.");

            register();

        }

    } catch (SQLException e) {

        e.printStackTrace();

    }

}

    public static void displayMainMenu() {

        boolean logout = false;

        while (!logout) {

            System.out.println("\n-- Main Menu --");

            System.out.println("1. Create new account");

            System.out.println("2. View existing accounts");

            System.out.println("3. Perform transaction");

            System.out.println("4. Logout");

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

            int choice = scanner.nextInt();

            scanner.nextLine();

            switch (choice) {

                case 1:

                    createNewAccount( );

                    break;

                case 2:

                    viewExistingAccounts();

                    break;

                case 3:

                    performTransactionMenu();

                    break;

                case 4:

                    logout();

                    logout = true;

                    break;

                default:

                    System.out.println("Invalid choice. Please enter a number between 1 and 4.");

            }

        }

    }

    public static void createNewAccount() {

        try (Connection conn = DriverManager.getConnection(DB\_URL, USER, PASS)) {

            System.out.println("\n===== Create New Account =====");

         Random random = new Random();

            int accountNumber = 1000000 + random.nextInt(9000000);

            System.out.println("Generated Account Number: " + accountNumber);

            System.out.print("Enter starting balance: ");

            double balance = scanner.nextDouble();

            scanner.nextLine();

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

            String username = scanner.nextLine();

            System.out.println("Choose account type:");

            System.out.println("1. Savings Account");

            System.out.println("2. Checking Account");

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

            int accountTypeChoice = scanner.nextInt();

            scanner.nextLine();

            String accountType = "";

            switch (accountTypeChoice) {

                case 1:

                    accountType = "Savings";

                    break;

                case 2:

                    accountType = "Checking";

                    break;

                default:

                    System.out.println("Invalid choice. Defaulting to Savings Account.");

                    accountType = "Savings";

            }

            String query = "INSERT INTO accounts (account\_number, balance, username,account\_type) VALUES (?, ?, ?, ?)";

            try (PreparedStatement preparedStatement = conn.prepareStatement(query)) {

                preparedStatement.setInt(1, accountNumber);

                preparedStatement.setDouble(2, balance);

                preparedStatement.setString(3,username);

                preparedStatement.setString(4, accountType);

                int rowsInserted = preparedStatement.executeUpdate();

                if (rowsInserted > 0) {

                    System.out.println("Account created successfully.");

                } else {

                System.out.println("Account creation failed.");

            }

        }

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

    public static void viewExistingAccounts() {

        try {

            String query = "SELECT \* FROM accounts WHERE username = ?";

            PreparedStatement preparedStatement = connection.prepareStatement(query);

            preparedStatement.setString(1, currentUser);

            ResultSet resultSet = preparedStatement.executeQuery();

            System.out.println(" ");

            System.out.println("-------------Existing accounts associated with the current user-------------");

            boolean found = false;

            while (resultSet.next()) {

                System.out.println("Account Number: " + resultSet.getInt("account\_number"));

                System.out.println("Balance: R" + resultSet.getDouble("balance"));

                System.out.println("Username: " + resultSet.getString("username"));

                System.out.println("Account Type: " + resultSet.getString("account\_type"));

                System.out.println("-----------------------------------------");

                found = true;

            }

               if (!found) {

                System.out.println("No accounts found for the current user.");

            }

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

    public static void insertTransactionRecord( String username,double amount, String transactionType) {

        try {

            String query = "INSERT INTO transactions (username, amount, transaction\_type) VALUES (?, ?, ?)";

            PreparedStatement preparedStatement = connection.prepareStatement(query);

            preparedStatement.setString(1, username);

            preparedStatement.setDouble(2, amount);

            preparedStatement.setString(3, transactionType);

           int rowsInserted = preparedStatement.executeUpdate();

           if (rowsInserted > 0) {

               System.out.println("Transaction recorded successfully.");

           } else {

               System.out.println("Failed to record transaction.");

           }

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

//reference: https://www.javatpoint.com/banking-application-in-java

    public static void performTransactionMenu() {

        System.out.println("\n-- Perform Transaction --");

        System.out.println("1. Perform transaction with Savings Account");

        System.out.println("2. Perform transaction with Checking Account");

        System.out.println("3. Go back to main menu");

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

        int choice = 0;

        try {

            choice = scanner.nextInt();

        } catch (Exception e) {

            System.out.println("Invalid input. Please enter a number.");

            scanner.next();

            return;

        }

        scanner.nextLine();

        switch (choice) {

            case 1:

                System.out.println("Performing transaction with Savings Account...");

              performTransaction(currentUser, "Savings");

                break;

            case 2:

                System.out.println("Performing transaction with Checking Account...");

                performTransaction(currentUser, "Checking");

                break;

            case 3:

                System.out.println("Going back to main menu.");

                break;

            default:

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

                break;

        }

    }

    public static void logout() {

        if (currentUser != null) {

            System.out.println("Logout successful. Goodbye, " + currentUser + "!");

            currentUser = null;

        } else {

            System.out.println("No user is currently logged in.");

        }

    }

    public static void performTransaction(Account account) {

        System.out.println("\n===== Perform Transaction =====");

        account.displayAccountInfo();

        System.out.println("1. Deposit");

        System.out.println("2. Withdraw");

        System.out.println("3. Transfer Funds");

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

        int choice = scanner.nextInt();

        scanner.nextLine();

        switch (choice) {

            case 1:

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

                final double depositAmount;

                try {

                    depositAmount = scanner.nextDouble();

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

                } catch (Exception e) {

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

                    scanner.next();

                    return;

                }

                scanner.nextLine();

                new Thread(new Runnable() {

                    @Override

                    public void run() {

                        try {

                            account.deposit(depositAmount);

                            insertTransactionRecord(currentUser, depositAmount, "Deposit");

                            updateAccountBalance(currentUser, account.getAccountNumber(), depositAmount);

                        } catch (IllegalArgumentException e) {

                            System.out.println("Invalid deposit amount: " + e.getMessage());

                        }

                    }

                }).start();

                break;

            case 2:

                System.out.print("Enter amount to withdraw: R");

                final double withdrawAmount;

                try {

                    withdrawAmount = scanner.nextDouble();

                } catch (Exception e) {

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

                    scanner.next();

                    return;

                }

                scanner.nextLine();

                //Reference: <https://www.callicoder.com/java-multithreading-thread-and-runnable-tutorial/>

//reference: https://www.javatpoint.com/runnable-interface-in-java

                new Thread(new Runnable() {

                    @Override

                    public void run() {

                        try {

                            if (account.withdraw(withdrawAmount)) {

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

                                insertTransactionRecord(currentUser, -withdrawAmount, "Withdrawal");

                               updateAccountBalance(currentUser, account.getAccountNumber(), -withdrawAmount);

                            } else {

                                System.out.println("Withdrawal failed: Insufficient funds.");

                            }

                        } catch (IllegalArgumentException e) {

                            System.out.println("Invalid withdrawal amount: " + e.getMessage());

                        }

                    }

                }).start();

                break;

            case 3:

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

            double transferAmount;

          System.out.print("Enter the amount to transfer: R");

            transferAmount = scanner.nextDouble();

            scanner.nextLine();

            transferFunds(currentUser, transferAmount);

            insertTransactionRecord(currentUser, transferAmount, "TRANSFER");

                break;

            default:

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

                break;

            }

        }

//Reference: https://www.shecodes.io/athena/54811-how-to-update-data-in-a-database-using-jsp

        public static void updateAccountBalance(String currentUser, String accountNumber, double amount) {

            try {

                String query = "UPDATE accounts SET balance = balance + ? WHERE username = ? AND account\_number = ?";

//reference: https://www.geeksforgeeks.org/how-to-use-preparedstatement-in-java/#:~:text=A%20PreparedStatement%20is%20a%20pre,to%20execute%20a%20parameterized%20query.

                PreparedStatement preparedStatement = connection.prepareStatement(query);

                preparedStatement.setDouble(1, amount);

                preparedStatement.setString(2, currentUser);

                preparedStatement.setString(3, accountNumber);

                int rowsUpdated = preparedStatement.executeUpdate();

                if (rowsUpdated > 0) {

                    System.out.println("Account balance updated successfully.");

                } else {

                    System.out.println("Failed to update account balance.");

                }

            } catch (SQLException e) {

                e.printStackTrace();

            }

        }

    public static void transferFunds(String currentUser,double transferAmount) {

        System.out.print("Enter the account number to transfer funds to: ");

        String receiverAccountNumber = scanner.nextLine();

        scanner.nextLine();

        try {

            String checkReceiverQuery = "SELECT \* FROM accounts WHERE account\_number = ?";

            PreparedStatement checkReceiverStatement = connection.prepareStatement(checkReceiverQuery);

            checkReceiverStatement.setString(1, receiverAccountNumber);

            ResultSet receiverResult = checkReceiverStatement.executeQuery();

            if (receiverResult.next()) {

                double senderBalance = getAccountBalance(currentUser);

                if (senderBalance >= transferAmount) {

                    String updateSenderBalanceQuery = "UPDATE accounts SET balance = balance - ? WHERE username = ?";

                    PreparedStatement updateSenderStatement = connection.prepareStatement(updateSenderBalanceQuery);

                    updateSenderStatement.setDouble(1, transferAmount);

                    updateSenderStatement.setString(2, currentUser);

                    updateSenderStatement.executeUpdate();

                    String updateReceiverBalanceQuery = "UPDATE accounts SET balance = balance + ? WHERE account\_number = ?";

                    PreparedStatement updateReceiverStatement = connection.prepareStatement(updateReceiverBalanceQuery);

                    updateReceiverStatement.setDouble(1, transferAmount);

                    updateReceiverStatement.setString(2, receiverAccountNumber);

                    updateReceiverStatement.executeUpdate();

                   insertTransactionRecord(currentUser, -transferAmount, "Transfer to " + receiverAccountNumber);

                   insertTransactionRecord(receiverResult.getString("username"), transferAmount, "Transfer from " + currentUser);

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

                } else {

                    System.out.println("Transfer failed: Insufficient balance.");

                }

            } else {

                System.out.println("Receiver account not found.");

            }

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

    public static double getAccountBalance(String username) {

        double balance = 0.0;

        try {

            String query = "SELECT balance FROM accounts WHERE username = ?";

            PreparedStatement preparedStatement = connection.prepareStatement(query);

            preparedStatement.setString(1, username);

            ResultSet resultSet = preparedStatement.executeQuery();

            if (resultSet.next()) {

                balance = resultSet.getDouble("balance");

            }

        } catch (SQLException e) {

            e.printStackTrace();

        }

        return balance;

    }

    public static void performTransaction(String username, String accountType) {

        try {

            String query = "SELECT \* FROM accounts WHERE username = ? AND account\_type = ?";

            PreparedStatement preparedStatement = connection.prepareStatement(query);

            preparedStatement.setString(1, username);

            preparedStatement.setString(2, accountType);

            ResultSet resultSet = preparedStatement.executeQuery();

            if (resultSet.next()) {

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

                String accountNumber = resultSet.getString("account\_number");

                Account account = null;

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

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

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

                    account = new CheckingAccount(username, accountNumber, balance, 0.0);

                }

                if (account != null) {

                    performTransaction(account);

                } else {

                    System.out.println("Error: Account type not recognized.");

                }

            } else {

                System.out.println("Error: No " + accountType + " account found for user " + username);

            }

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

        }

    //BankingOperation.java

public interface BankingOperations {

    double checkBalance();

    void deposit(double amount);

    boolean withdraw(double amount);

   void transferFunds(Account toAccount, double amount);

}

//BankAccount.java

public abstract class BankAccount implements BankingOperations {

    protected String accountHolder;

    protected String accountNumber;

    protected double balance;

    public BankAccount(String accountHolder, String accountNumber, double balance) {

        this.accountHolder = accountHolder;

        this.accountNumber = accountNumber;

        this.balance = balance;

    }

    @Override

    public double checkBalance() {

        return balance;

    }

    @Override

    public void deposit(double amount) {

        balance += amount;

    }

    @Override

    public boolean withdraw(double amount) {

        if (balance >= amount) {

            balance -= amount;

            return true;

        }

        return false;

    }

}

//Account.java

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.SQLException;

public class Account {

    protected String accountHolder;

    protected String accountNumber;

    protected double balance;

    private static double transactionFee = 0.5;

    private static double interestRate = 0.5;

    private Connection connection;

    private String username;

    public static double getTransactionFee() {

        return transactionFee;

    }

    public static void setTransactionFee(double fee) {

        transactionFee = fee;

    }

    public static double getInterestRate() {

        return interestRate;

    }

    public static void setInterestRate(double rate) {

        interestRate = rate;

    }

    public Account(String accountHolder, String accountNumber, double balance) {

        this.accountHolder = accountHolder;

        this.accountNumber = accountNumber;

        this.balance = balance;

    }

    public String getAccountHolder() {

        return accountHolder;

    }

    public String getAccountNumber() {

        return accountNumber;

    }

    public double getBalance() {

        return balance;

    }

    public void deposit(double amount) {

        balance += amount;

    }

    public boolean withdraw(double amount) {

        if (balance >= amount) {

            balance -= amount;

            return true;

        }

        return false;

    }

    public void displayAccountInfo() {

        System.out.println("Account Holder: " + accountHolder);

        System.out.println("Account Number: " + accountNumber);

        System.out.println("Current Balance: R" + balance);

    }

    public String getUsername() {

       return this.username;

    }

}

class SavingsAccount extends Account {

    public SavingsAccount(String accountHolder, String accountNumber, double balance) {

        super(accountHolder, accountNumber, balance);

    }

    @Override

    public boolean withdraw(double amount) {

        if (balance >= amount) {

            balance -= amount;

            return true;

        }

        System.out.println("Withdrawal failed: Insufficient funds.");

        return false;

    }

    public double checkBalance() {

        return balance;

    }

    public class InsufficientFundsException extends RuntimeException {

        public InsufficientFundsException(String message) {

            super(message);

        }

    }

}

class CheckingAccount extends Account {

    private double overdraftLimit;

    public CheckingAccount(String accountHolder, String accountNumber, double balance, double overdraftLimit) {

        super(accountHolder, accountNumber, balance);

        this.overdraftLimit = overdraftLimit;

    }

    public double getOverdraftLimit() {

        return overdraftLimit;

    }

    @Override

    public boolean withdraw(double amount) {

        if (balance + overdraftLimit >= amount) {

            balance -= amount;

            return true;

        }

        System.out.println("Withdrawal failed: Insufficient funds.");

        return false;

    }

}