

**Subject: Object Oriented Programming**

**Subject code: IT602**

**Date of submission:5-2-2022**

**Due date of submission:5-2-2022**

**Group Name: Elite**

**Group No: 3**

**Teaching Asst: Rutvik Prajapati**

**Student Names:**

**Dev Adnani 202212012**

**Rudrik Prajapati 202212013**

**Sanket Acharya 202212100**

**Yash Khatri 202212086**

**Topic: Develop a console-based application for a retailstore.**

**Question Statement**

**Problem Statement:** Construct a console-based application for a **banking system application**.

**Part I: User Login/ Register**

* Application should start by showing two options: (1) Sign In (2) Register
* In the case of Sign In, ask the user to provide his “username” and “password”. If they match, let her in, not otherwise.
* In the case of Register, ask the user to provide his “full name”, “username”, and “password”. In case of password, echo *asterisk* symbol (\*) in place of the typed character. E.g., a user types “1@4”, you should display “\*\*\*”, but you will store the actual password to match when she logs in again.

**Hint:** You can use **java.io.Console.readPassword()** method. Carefully read its description from the Java 8 docs online.

* In case of Sign In, provide an option of “Forgot Password”. Ask the user for her name, if it matches then, let her set a new password and redirect her to login again with the new password.

**Part II : Assignment 2**

* In the assignment 1 , while opening the account of the user please take one more input of type of account ( saving or current ) .
* Define a interface named Calculate\_interest with abstract method to calculate interest of saving account. Take input for the user amount, time and rate is 5%. Make this method private also. Formula: (Amount\*time\*rate)/100. And display the interest for the amount and total balance.
* Define one more interface named Approve\_Loan with two abstract methods.
* 1: For saving account – if balance is more than 1 lakh then approved loan of amount of (1.5\* balance). And display that the loan is approved of this amount or not approved.
* 2: For current account – if balance is more than 5 lakh then approved loan of amount of (3\* balance).
* And display that the loan is approved of this amount or not approved.

**Part III : Transfer money from one account to another**

* Add an admin account which can only read all the information of other accounts.(account no. & balance).
* Make an interface which provides an option to transfer money from one account to another.
* At the end show the details of all the accounts (account no. – balance) through the admin.

**File Structure**

src

* account(folder/package)
  + Account.java
  + AccountType.java
  + CurrentAccount.java
  + SavingsAccount.java
* accountHelper(folder/package)
  + AccountHelper.java
* admin(folder/package)
  + Admin.java
  + AdminInterface.java
* adminHelper(folder/package)
  + AdminHelper.java
* bank Operations(folder/package)
  + ApproveLoan.java
  + CalculateInterest.java
  + Transferable
* utils(folder/package)
  + Utils.java

main.java

Account Package :

**Account.java**

package account;

import bankOperations.ApproveLoan;

import bankOperations.CalculateInterest;

import static utils.Utils.\*;

public class Account implements CalculateInterest, ApproveLoan {

private static int *lastAccountNo* = 0;

private String username, password, fullName;

private double balance = 0;

private int accountNumber;

private double minimumBalance;

private AccountType accountType;

private double interestRate;

// Constructor

public Account(String username, String fullName) {

this.username = username;

this.fullName = fullName;

this.accountNumber = ++*lastAccountNo*;

}

public void interest(double amount, int years) {

double interestAmount = (amount \* years \* interestRate) / 100;

System.*out*.println("You will get an interest of " + interestAmount + " each year !");

}

//Get Account Type

public AccountType getAccountType() {

return accountType;

}

//Set Account Type

public void setAccountType(AccountType accountType) {

this.accountType = accountType;

}

//Get Interest Rate

public double getInterestRate() {

return interestRate;

}

//Set Interest Rate

public void setInterestRate(double interestRate) {

this.interestRate = interestRate;

}

// Get - FullName

public String getFullName() {

return fullName;

}

// Set - FullName

public void setFullName(String fullName) {

this.fullName = fullName;

}

//Get - UserName

public String getUsername() {

return username;

}

// Set Username

public void setUsername(String username) {

this.username = username;

}

// Get - Password

public String getPassword() {

return password;

}

// Set - Password

public void setPassword(String password) {

this.password = password;

}

// Get User ID

public int getAccountNumber() {

return accountNumber;

}

// Get Balance

public double getBalance() {

return balance;

}

// Set Balance

public void setBalance(double balance) {

this.balance = balance;

}

//Get MinimumBalance

public double getMinimumBalance() {

return minimumBalance;

}

//Set MinimumBalance

public void setMinimumBalance(double minimumBalance) {

this.minimumBalance = minimumBalance;

}

//Withdrawal From User

public boolean withdrawAmountUser(double withdrawalAmt) {

if (withdrawalAmt <= 0 ||withdrawalAmt > balance || balance - getMinimumBalance() < withdrawalAmt) {

return false;

}

else {

setBalance(getBalance() - withdrawalAmt);

return true;

}

}

//Deposit Amount From User

public void depositAmountUser(double enterAmount,int noOfYears) {

setBalance(getBalance() + enterAmount);

interest(getBalance(), noOfYears);

}

public void deposit(double amount) {

setBalance(getBalance() + amount);

}

// Withdraw method

public void withdraw(double amount) {

if (balance >= amount) balance -= amount;

}

@Override

public void approveOnSavings(double loanAmt) {

}

@Override

public void approveOnCurrent(double loanAmt) {

}

}

Description :

* This is a Java class named "Account" in the "account" package.
* It implements two interfaces, "CalculateInterest" and "ApproveLoan", and imports classes "ApproveLoan" and "CalculateInterest" from the "bankOperations" package.
* It also imports the "utils" package and the static method "Utils" from it.
* The class has private fields such as username, password, fullName, balance, accountNumber, minimumBalance, accountType, and interestRate.
* The lastAccountNo variable is a static field, which keeps track of the last account number assigned to a new account.
* The constructor initializes the username, full name, and account number.
* The class has several public methods such as interest, getFullName, setFullName, getUsername, setUsername, getPassword, setPassword, getAccountNumber, getBalance, setBalance, getMinimumBalance, setMinimumBalance, withdrawAmountUser, depositAmountUser, deposit, and withdraw.
* These methods allow users to perform different operations on an account object such as checking the interest rate, getting the full name of the account holder, setting a new password, depositing or withdrawing money, etc.
* The class also has two empty method implementations, approveOnSavings and approveOnCurrent, as it implements the ApproveLoan interface.
* Overall, this class provides a basic implementation of an account with various features that are typical of a bank account.

Account Package :

**Account.java**

package account;

public enum AccountType {

*SAVINGS\_ACCOUNT*, *CURRENT\_ACCOUNT*

}

Description :

* This is a Java enumeration named "AccountType" in the "account" package.
* It defines two constant values, "SAVINGS\_ACCOUNT" and "CURRENT\_ACCOUNT", which represent different types of accounts.
* This enumeration is used to specify the type of account in the "Account" class.

Account Package :

**CurrentAccount.java**

package account;

public class CurrentAccount extends Account {

public CurrentAccount(String fullName, String userName) {

super(fullName, userName);

setAccountType(AccountType.*CURRENT\_ACCOUNT*);

setInterestRate(2);

setMinimumBalance(5000);

}

@Override

public void approveOnCurrent(double loanAmt) {

double balance = getBalance();

if (balance >= 500000) {

if (loanAmt > 0 && loanAmt <= balance \* 3) {

System.*out*.println("Your Loan is Approved!");

} else {

System.*out*.println("Your request has been noted. Your branch will contact you for followup!");

}

} else {

System.*out*.println("Your request has been noted. Your branch will contact you for followup!");

}

}

}

Description :

* This is a Java class named "CurrentAccount" in the "account" package.
* This class extends the "Account" class, which means it inherits all the fields and methods of the "Account" class and can also have its own fields and methods.
* The constructor takes two parameters, "fullName" and "userName", which are passed to the constructor of the superclass "Account" using the "super" keyword.
* The constructor then sets the account type to "CURRENT\_ACCOUNT", interest rate to 2, and minimum balance to 5000 using the setter methods of the superclass.
* This class overrides the "approveOnCurrent" method of the "ApproveLoan" interface, which was implemented by the "Account" class.
* The "approveOnCurrent" method checks if the balance of the account is greater than or equal to 500000.
* This class can be used to create current account objects that have specific properties, such as a minimum balance and interest rate, and can approve loans based on specific conditions.

Account Package :

**SavingsAccount.java**

package account;

public class SavingsAccount extends Account {

public SavingsAccount(String fullName, String userName) {

super(fullName, userName);

setAccountType(AccountType.*SAVINGS\_ACCOUNT*);

setInterestRate(7);

setMinimumBalance(1000);

}

@Override

public void approveOnSavings(double loanAmt) {

double balance = getBalance();

if (balance >= 100000) {

if (loanAmt > 0 && loanAmt <= balance \* 1.5) {

System.*out*.println("Your Loan is Approved!");

} else {

System.*out*.println("Your request has been noted. Your branch will contact you for followup!");

}

} else {

System.*out*.println("Your request has been noted. Your branch will contact you for followup!");

}

}

}

Description :

* The SavingsAccount class has a constructor that takes in two parameters, fullName and userName, which are passed to the constructor of the Account superclass using the super keyword.
* The constructor also sets the account type to SAVINGS\_ACCOUNT, interest rate to 7 and the minimum balance to 1000.
* The class overrides the approveOnSavings method of the Account class.
* The approveOnSavings method takes a loanAmt parameter and checks whether the balance in the account is greater than or equal to 100000. If the balance is greater than or equal to 100000, it checks whether the requested loan amount is greater than zero and less than or equal to 1.5 times the balance.

AccountHelper Package :

**AccountHelper.java**

package accountHelper;

import account.Account;

import account.CurrentAccount;

import account.SavingsAccount;

import java.util.HashMap;

import static utils.Utils.\*;

public class AccountHelper {

public Account createAccount(String accountType, String fullName, String userName) {

if (accountType.equalsIgnoreCase("savings")) {

return new SavingsAccount(fullName, userName);

} else {

return new CurrentAccount(fullName, userName);

}

}

//Forget Password - Just Checking If Name Is Same As Old Or Not , If Yes We Change The Password , Else We Return False

public boolean forgetPassword(HashMap<String, Account> users) {

String userName, password;

userName = *userInputString*("Username");

password = *hashedPassword*();

boolean userNameCheck = users.containsKey(userName);

if (userNameCheck) {

users.get(userName).setPassword(password);

return true;

} else {

return false;

}

}

public boolean registerUser(HashMap<String, Account> users) {

String userName, password, fullName, isMember;

boolean isRegistered = false;

try {

fullName = *userInputString*("Full name");

userName = *userInputString*("Username");

isMember = *userInputString*("Yes For Savings Account ,No For Current Account : ");

password = *hashedPassword*();

if (fullName.isEmpty() || userName.isEmpty() || isMember.isEmpty() || password.isEmpty()) {

*print*("Please Enter Some Data !xD");

} else {

//Checking If Username Already Exists - Not Permitting Multiple Users With Same Username

boolean allowUser = users.containsKey(userName);

if (!allowUser) {

Account account = createAccount(isMember.equalsIgnoreCase("yes") ? "savings" : "current", fullName, userName);

account.setPassword(password);

users.put(userName, account);

isRegistered = true;

}

}

} catch (Exception e) {

*print*("An error occurred while registering user: " + e.getMessage());

}

return isRegistered;

}

public Account loginUser(HashMap<String, Account> users) {

String userName, password;

Account loggedUser = null;

try {

userName = *userInputString*("Username");

boolean userNameCheck = users.containsKey(userName);

if (userNameCheck) {

password = *hashedPassword*();

String userPass = users.get(userName).getPassword();

if (userPass.equals(password)) {

loggedUser = users.get(userName);

}

}

} catch (Exception e) {

*print*("An error occurred while logging in: " + e.getMessage());

}

return loggedUser;

}

public void displayUserProfile(Account user) {

try {

*print*("--------------------------- USER PROFILE-------------------------------- ");

*print*("Account Number : " + user.getAccountNumber());

*print*("Full Name : " + user.getFullName());

*print*("Username : " + user.getUsername());

*print*("Balance : " + user.getBalance());

*print*("Account Type : " + user.getAccountType());

*print*("Interest Rate : " + user.getInterestRate() + "%");

*print*("----------------------END OF USER PROFILE-------------------------------- ");

} catch (Exception e) {

*print*("An error occurred while displaying user profile: " + e.getMessage());

}

}

}

Description :

* This code defines a class called AccountHelper that provides various helper methods to create, manage, and display user accounts.

The methods provided are:

* createAccount: This method creates a new account of the specified type ("savings" or "current"), with the given full name and username, and returns the account object.
* forgetPassword: This method checks if the given username exists in the users hash map and if yes, changes the password to the newly entered password and returns true. Otherwise, it returns false.
* registerUser: This method registers a new user by taking in their full name, username, account type ("savings" or "current"), and password. It then checks if the given username already exists and if not, creates a new account and adds it to the users hash map with the given username as the key. It returns true if the user was successfully registered, otherwise false.
* loginUser: This method takes in a username and password, checks if the given username exists in the users hash map and if yes, checks if the entered password matches the password for the account with that username. If the passwords match, it returns the account object for that user, otherwise it returns null.
* displayUserProfile: This method takes in an account object and displays the user's account number, full name, username, balance, account type, and interest rate.
* The AccountHelper class depends on two other classes called Account and its subclasses SavingsAccount and CurrentAccount, which are not defined in this code snippet.

Admin Package :

**Admin.java**

package admin;

import account.Account;

import accountHelper.AccountHelper;

import bankOperations.Transferable;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

public class Admin implements AdminInterface, Transferable {

AccountHelper accountHelper = new AccountHelper();

@Override

public void checkAllAccounts(HashMap<String, Account> users) {

users.forEach((key, account) -> {

accountHelper.displayUserProfile(account);

});

}

@Override

public void getDataOfUserAccount(Account account) {

accountHelper.displayUserProfile(account);

}

@Override

public void writeDataToFile(HashMap<String, Account> users) {

ExecutorService executor = Executors.*newFixedThreadPool*(2);

// write the contents of the HashMap to a file

try (BufferedWriter writer = new BufferedWriter(new FileWriter("output.txt"))) {

for (Map.Entry<String, Account> entry : users.entrySet()) {

// submit a task to the executor to write the entry to the file

executor.submit(() -> {

try {

writer.write(entry.getKey() + "=" + entry.getValue().toString() + "\n");

} catch (IOException e) {

e.printStackTrace();

}

});

}

// wait for all tasks to complete

executor.shutdown();

executor.awaitTermination(Long.*MAX\_VALUE*, java.util.concurrent.TimeUnit.*NANOSECONDS*);

} catch (IOException e) {

e.printStackTrace();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

@Override

public boolean transfer(Account sourceAccount, Account destAccount, double amount) {

if (sourceAccount.getBalance() >= amount) {

sourceAccount.withdraw(amount);

destAccount.deposit(amount);

return true;

} else {

return false;

}

}

}

Description :

* This is a Java class named Admin that implements two interfaces: AdminInterface and Transferable.
* The Admin class has a reference to an object of the AccountHelper class, which is used to perform various account-related operations.
* The checkAllAccounts() method accepts a HashMap of users and iterates over each user's account to display their profile using the displayUserProfile() method of the AccountHelper class.
* The getDataOfUserAccount() method displays the profile of a given account using the displayUserProfile() method of the AccountHelper class.
* The transfer() method transfers a given amount from a source account to a destination account. If the source account has sufficient balance, the transfer is made, and the method returns true. Otherwise, the transfer fails, and the method returns false.

Admin Package :

**AdminInterface.java**

package admin;

import account.Account;

import java.util.HashMap;

public interface AdminInterface {

public void checkAllAccounts(HashMap<String, Account> users);

public void getDataOfUserAccount(Account account);

}

Description :

This is an interface named AdminInterface which is used in the Admin class. It contains two methods:

* checkAllAccounts(HashMap<String, Account> users): This method takes a HashMap of String and Account objects as input and displays the profile of all the user accounts in the hashmap.
* getDataOfUserAccount(Account account): This method takes an Account object as input and displays the profile of the user account.

AdminHelper Package :

**AdminHelper.java**

package adminHelper;

import account.Account;

import admin.Admin;

import java.util.HashMap;

import static utils.Utils.\*;

public class AdminHelper {

boolean isLogged = false;

Admin admin = new Admin();

public void showAdminMenu(HashMap<String, Account> users) {

while (true) {

try {

while (!isLogged) {

*print*("Enter 1 For Login");

*print*("Enter 2 For Exit");

int ch = *userInputInt*("Enter Your Choice :");

switch (ch) {

case 1 -> {

String uname = *userInputString*("username");

String password = *hashedPassword*();

if (uname.equals("admin") && password.equals("daiict")) {

isLogged = true;

} else {

throw new Exception("Invalid username or password");

}

}

case 2 -> {

return;

}

default -> {

throw new Exception("Invalid choice, please enter a valid choice");

}

}

}

while (true) {

*print*("Enter 1 For Getting All User List");

*print*("Enter 2 For Getting Detail Of Particular User");

*print*("Enter 3 For Transferring Money Between Users");

*print*("Enter 4 Going Back To Main Screen");

*print*("Enter 5 For Clearing Screen");

*print*("Enter 6 For Generating File of All Account");

int ch = *userInputInt*("Enter Your Choice :");

switch (ch) {

case 1 -> {

admin.checkAllAccounts(users);

}

case 2 -> {

String userName = *userInputString*("Username");

if (userName.isEmpty()) {

throw new Exception("Please Enter Username");

} else if (!users.containsKey(userName)) {

throw new Exception("User doesn't exist");

} else {

Account account = users.get(userName);

admin.getDataOfUserAccount(account);

}

}

case 3 ->

{

String source = *userInputString*("Username of source account");

String dest = *userInputString*("Username of dest account");

double amt = *userInputDouble*("Enter The Amount To Be Transferred : ");

if (source.isEmpty() || dest.isEmpty()) {

throw new Exception("Please Enter Username");

} else if (!users.containsKey(source)) {

throw new Exception("Source user doesn't exist");

} else if (!users.containsKey(dest)) {

throw new Exception("Destination user doesn't exist");

} else {

Account sourceAccount = users.get(source);

Account destAccount = users.get(dest);

boolean transferred = admin.transfer(sourceAccount,destAccount,amt);

if(transferred)

*print*("Amount Transferred Successfully");

else

*print*("Oops Some Error Occurred");

}

}

case 4 -> {

isLogged = false;

return;

}

case 5 -> *clearTerminal*();

default -> {

throw new Exception("Invalid choice, please enter a valid choice");

}

case 6 ->

{

admin.writeDataToFile(users);

}

}

}

} catch (Exception e) {

*print*("Error occurred: Enter Valid Data");

}

}

}

}

Description :

* This is a Java class named AdminHelper that provides methods to display a menu of options to an admin user and perform different operations related to banking. Here's a description of the class and its methods:
* The class has a boolean field named isLogged that indicates whether the admin user is currently logged in or not.
* The class has an instance variable of the Admin class named admin. This allows the AdminHelper to access the methods provided by the Admin class.
* The class has a method named showAdminMenu that takes a HashMap of user accounts as input and displays a menu of options to the admin user. The method uses a while loop to keep displaying the menu until the user chooses to exit.
* The method first displays a login menu that allows the admin user to log in by entering a username and password. The username is hardcoded as "admin" and the password is hardcoded as "daiict". If the user enters incorrect credentials, an exception is thrown and the user is prompted to enter valid credentials.
* After the user logs in, the method displays the main menu with the following options:
  + Option 1: Get all user accounts - This option calls the checkAllAccounts method of the Admin class to display the details of all user accounts in the system.
  + Option 2: Get detail of particular user - This option prompts the admin user to enter the username of a particular user account and calls the getDataOfUserAccount method of the Admin class to display the details of that user account.
  + Option 3: Transfer money between users - This option prompts the admin user to enter the usernames of the source and destination accounts and the amount to be transferred. The method then calls the transfer method of the Admin class to perform the transfer operation. If the transfer is successful, a success message is displayed. Otherwise, an error message is displayed.
  + Option 4: Go back to the main screen - This option returns to the main menu.
  + Option 5: Clear screen - This option clears the terminal screen.
* If the user enters an invalid choice at any point, an exception is thrown and the user is prompted to enter a valid choice.
* If an exception is thrown at any point in the method, an error message is displayed and the user is prompted to enter valid data.

bankOperations Package :

**ApproveLoan.java**

package bankOperations;

public interface ApproveLoan {

void approveOnSavings(double loanAmt);

void approveOnCurrent(double loanAmt);

}

Description :

* The "bankOperations" package contains an interface named "ApproveLoan". This interface has two methods defined, namely "approveOnSavings" and "approveOnCurrent".

bankOperations Package :

**CalculateInterest.java**

package bankOperations;

public interface CalculateInterest {

private void interest(int amount,int time){

}

}

Description :

* The given code defines an interface CalculateInterest in the bankOperations package. This interface declares a private method interest that takes in two parameters amount and time, both of type int. However, private methods are not allowed to be declared inside an interface.

bankOperations Package :

**CalculateInterest.java**

package bankOperations;

import account.Account;

public interface Transferable {

// Transfers the specified amount from the source account to the destination account

public boolean transfer(Account sourceAccount, Account destAccount, double amount);

}

Description :

* The Transferable interface is defined in the bankOperations package. It defines a single method transfer that takes in two Account objects representing the source and destination accounts respectively, along with a double value amount representing the amount of money to be transferred from the source account to the destination account. The method returns a boolean value indicating whether or not the transfer was successful.

utils Package :

**Utlis.java**

package utils;

import java.io.Console;

import java.util.Scanner;

public class Utils {

public static void print(String msg) {

System.*out*.println(msg);

}

public static String hashedPassword() {

Console console = System.*console*();

String tempPass = "";

if (console != null) {

char tx[] = console.readPassword("Enter Password : ");

tempPass = String.*valueOf*(tx);

if (tempPass.length() != 0) {

for (char c : tx) {

System.*out*.print("\*");

}

}

*print*("");

} else {

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

*print*("Enter Password : ");

tempPass = scanner.nextLine();

}

return tempPass;

}

public static String userInputString(String t) {

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

String data = "";

*print*("Please Enter " + t + " : ");

data = scanner.nextLine();

return data;

}

public static int userInputInt(String t) {

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

int no;

*print*(t);

no = Integer.*parseInt*(scanner.nextLine());

return no;

}

public static double userInputDouble(String t) {

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

double no;

*print*(t);

no = Double.*parseDouble*(scanner.nextLine());

return no;

}

public static void clearTerminal() {

System.*out*.print("\033[H\033[2J");

System.*out*.flush();

}

}

Description :

* The utils package contains utility classes and methods that are commonly used in various parts of the banking application.
* The Utils class contains several static methods that perform common operations such as printing messages to the console, reading input from the user, and clearing the console.
* The print method simply prints a message to the console.
* The hashedPassword method prompts the user to enter a password securely, using the Console class if available, and masking the password with asterisks.
* The userInputString, userInputInt, and userInputDouble methods prompt the user to enter data of the specified type and return the entered value.
* The clearTerminal method clears the console by printing a special sequence of characters.

**Main.java**

import account.Account;

import account.AccountType;

import accountHelper.AccountHelper;

import admin.Admin;

import adminHelper.AdminHelper;

import java.util.HashMap;

import static java.lang.System.*exit*;

import static utils.Utils.\*;

public class Main {

//Note : Please Compile And Run Code In Terminal As We Have Used Console (For Password) , Which Only Works In Terminal

public static void main(String[] args) {

boolean isRegistered = false, isLoggedIn = false;

AccountHelper accountHelper = new AccountHelper();

HashMap<String, Account> users = new HashMap<>();

AdminHelper adminHelper = new AdminHelper();

Account loggedUser = null;

int choice;

while (true) {

while (!isLoggedIn || !isRegistered) {

*print*("Enter 1 For Registering");

*print*("Enter 2 For Login");

*print*("Enter 3 For Admin Access");

*print*("Enter 4 For Forget Password");

*print*("Enter 5 For Clearing Screen");

*print*("Enter 6 For Closing The App");

try {

choice = *userInputInt*("Enter Your Choice");

switch (choice) {

case 1: {

isRegistered = accountHelper.registerUser(users);

if (!isRegistered) {

*print*("Username Already Exists");

}

break;

}

case 2: {

if (isRegistered) {

loggedUser = accountHelper.loginUser(users);

if (loggedUser == null) {

*print*("Wrong username or Password");

} else {

isLoggedIn = true;

}

} else {

*print*("Please Register Before Trying To Login");

}

break;

}

case 3: {

adminHelper.showAdminMenu(users);

}

case 4: {

if (isRegistered) {

boolean passChangeStatus = accountHelper.forgetPassword(users);

if (passChangeStatus) {

*print*("Password Changed Successfully");

} else {

*print*("Oops Wrong Username");

}

} else {

*print*("You Must Register Before , If You Wanna Change Password");

}

break;

}

case 5: {

*clearTerminal*();

break;

}

case 6: {

*print*("Bye Bye");

*exit*(1);

break;

}

default:

*print*("Wrong Choice...");

break;

}

} catch (NumberFormatException e) {

*print*("Invalid Input! Please enter a valid number.");

} catch (Exception e) {

*print*("An error occurred: " + e.getMessage());

}

}

while (true) {

*print*("Enter 1 For User Profile");

*print*("Enter 2 For Deposit");

*print*("Enter 3 For Withdrawal");

*print*("Enter 4 For Loan");

*print*("Enter 5 For Transferring ");

*print*("Enter 6 For Logout");

*print*("Enter 7 For Closing The App");

*print*("Enter 8 For Clearing Screen");

try {

choice = *userInputInt*("Enter Your Choice");

switch (choice) {

case 1:

accountHelper.displayUserProfile(loggedUser);

break;

case 2: {

double enterAmount = *userInputDouble*("Enter Amount To Deposit : ");

int noOfYears = *userInputInt*("Enter Number Of Years :");

if (enterAmount <= 0 || noOfYears < 0) {

*print*("Invalid choice");

return;

} else {

loggedUser.depositAmountUser(enterAmount, noOfYears);

*print*("Deposit successfully!");

}

break;

}

case 3: {

double withdrawalAmt = *userInputDouble*("Enter the amount you want to withdraw :");

boolean withdrawalDone = loggedUser.withdrawAmountUser(withdrawalAmt);

if (withdrawalDone)

*print*("Withdraw successful!");

else {

*print*("Please check the amount");

*print*("Note : The Minimum Balance Should Be" + loggedUser.getMinimumBalance());

}

break;

}

case 4: {

double loanAmt = *userInputDouble*("Enter Amount For Loan");

if (loggedUser.getAccountType() == AccountType.*SAVINGS\_ACCOUNT*)

loggedUser.approveOnSavings(loanAmt);

else

loggedUser.approveOnCurrent(loanAmt);

break;

}

case 5:

{

String dest = *userInputString*("Username of dest account");

double amt = *userInputDouble*("Enter The Amount To Be Transferred : ");

if (dest.isEmpty()) {

throw new Exception("Please Enter Username");

} else if (!users.containsKey(dest)) {

throw new Exception("Destination user doesn't exist");

} else {

Account destAccount = users.get(dest);

boolean transferred = new Admin().transfer(loggedUser,destAccount,amt);

if(transferred)

*print*("Amount Transferred Successfully");

else

*print*("Oops Some Error Occurred");

}

}

break;

case 6:

*print*("Processing.");

*print*("Processing.....");

*print*("Processing.......");

*print*("Processing.........");

*print*("Processing...........");

break;

case 7:

*print*("Bye Bye");

*exit*(1);

break;

case 8:

*clearTerminal*();

break;

default:

*print*("Enter Valid Choice");

break;

}

if (choice == 6) {

*print*("Good Bye , Hope You Had A Good Time");

loggedUser = null;

isLoggedIn = false;

*clearTerminal*();

break;

}

} catch (NumberFormatException e) {

*print*("Invalid Input! Please enter a valid number.");

} catch (Exception e) {

*print*("An error occurred: " + e.getMessage());

}

}

}

}

}

Description :

* The given code is a Java program that implements a console-based banking application. The program allows users to register and login, and then perform various banking transactions. It also has an admin access feature.
* The program consists of several classes, including the Account and AccountType classes for representing user accounts and their types, the AccountHelper class for handling user account operations, and the Admin and AdminHelper classes for admin access and operations.
* The Main class is the main entry point for the program. It contains the main method that starts the program and handles user input and menu navigation. The Main class also contains a HashMap object called users for storing user accounts, and several boolean variables for tracking user login and registration status.
* The program's menu system is divided into two parts: the login/registration menu, and the main banking menu. The login/registration menu allows users to register or login, access admin functions, or forget their password. The main banking menu allows logged-in users to perform various banking transactions such as viewing their profile, depositing or withdrawing money, applying for a loan, transferring funds, or logging out.
* The program also contains several utility methods for handling console input/output and clearing the console screen. These include the print, userInputInt, userInputDouble, and userInputString methods, as well as the clearTerminal method.
* Overall, the program provides a simple, console-based banking application that demonstrates several core Java programming concepts, such as object-oriented design, collections, user input/output, and exception handling.

Screenshot :

