Suppose that a bank offers customers the following account types:

• A savings account that earns interest. The interest compounds monthly and is computed on the minimum monthly balance.

• A checking account that has no interest, gives you three free withdrawals per month, and charges a $1 transaction fee for each additional withdrawal.

Design and implement a program that will manage a set of accounts of both types. It should be structured so that other account types can be added without affecting the main processing loop.

Supply a menu :

D)eposit

W)ithdraw

M)onth end

S)ave

L)oad

Q)uit.

For deposits and withdrawals, query the account number and amount.

Print the balance of the account after each transaction.

In the “Month end” command, accumulate interest or clear the transaction counter, depending on the type of the bank account.

Then print the balance of all accounts.

The program should be apple to save the data to a text file and load them to the program when the user chooses the appropriate command from the menu.

**UML:**

***Expl :***

**Bank**.**java:**

The Bank class is a blueprint for creating bank objects.

It has three properties: name, address and swiftCode.

It has a constructor that takes three parameters: name, address and swiftCode.

It has getters and setters for all three properties.

It has a toString method that returns a string representation of the object.

It has a covertToCSVLine method that returns a string representation of the object in CSV format.

**Costumer.java:**

The code is a class that creates a costumer.

It has a costumer number, a first registration date, an id, a name, a surname, an email, a gender, if the costumer is married and a password.

It also has methods to get and set the values of the variables.

It also has methods to covert the values of the variables to CSV lines and to string.

**Account.java:**

The Account class is a blueprint for creating objects that represent a bank account.

The class has five fields, which are variables that store information about the object.

The first field is an integer called costumerNr. The second field is a String called accNr. The third field is a String called iban. The fourth field is a double called amount. The fifth field is a String called currency.

The class has one constructor, which is used to create objects from the class. The constructor has five parameters, which are used to initialize the fields of the object when it's created.

The class has two methods, which are functions that perform operations on the object's fields. The first method is deposit, which takes one parameter of type double and adds it to the amount field of the object that calls it. The second method is withdraw, which takes one parameter of type double and subtracts it from the amount field of the object that calls it.

**CheckingAccount :**

The CheckingAccount class is a subclass of the Account class.

The code is a class that extends the Account class.

It has a maintenanceFee variable.

It has a constructor that takes in the costumerId, accNr, iban, amount, currency and maintenanceFee.

It has getters and setters for the maintenanceFee variable.

It has a toString method that returns a string with the maintenanceFee variable and the superclass toString method.

It has a covertToCSVLine method that returns a string with the superclass covertToCSVLine method and the maintenanceFee variable.

**SavingsAccount:**

The SavingsAccount class is a subclass of the Account class.

It has a private double variable called interestRate.

It has a constructor that takes 6 parameters and passes the first 4 to the superclass constructor.

It has a getter and setter for interestRate.

It has a method called monthEnd that calculates the interest for one month and deposits it to the account.

It overrides the toString method from the superclass and adds information about interestRate.

It overrides the covertToCSVLine method from the superclass and adds information about interestRate.

**Transaction:**

The code is a class that is used to create objects of the type Transaction.

The class has 6 attributes, all of which are private.

The attributes are: transactionNr, costumerId, accountNr, time, transactionType and amount.

The attributes are of the following types: int, String, String, LocalDateTime, TransactionType and double.

The class has a constructor with 6 parameters. The parameters are: transactionNr, costumerId, accountNr, time, transactionType and amount. The parameters are of the following types: int, String, String, LocalDateTime, TransactionType and double. The constructor initializes the attributes with the values of the parameters.

The class has getters and setters for all attributes.

The class has a method called covertToCSVLine that returns a String value. The method does not take any parameters. The method returns a string that contains all attribute values separated by commas (",").

The **TransactionType** enum is a list of constants.

The enum constants are DEPOSIT, WITHDRAW, INTEREST\_INCOME, and WITHDRAW\_FEE.

**banka data :**

**The code is a class that contains the data of the bank.**

It has a constructor that calls the method reloadData() which sets all the data from the files.

It has getters and setters for all the data.

It has a method checkMaxTransNr() which checks what is the highest transaction number

in the transactions list.

It has a method readFileLines() which reads all lines from a file and returns them as a list of strings.

It has methods

setBanksFromFile(),

setSavingsAccountsFromFile(),

setCheckingAccountsFromFile(),

setCostumersFromFile() and

setTransactionsFromFile() which sets all the data from their respective files.

It has a method saveAllInFiles() which saves all data in their respective files.

It has a method reloadData() which calls all methods that set data from files.

**BankUI:**

The code is a program that simulates a bank.

**The program starts with the main method.**

The main method creates a new instance of the BankUI class and calls the startProgram method on it.

The program has a menu where you can choose to login or quit the program.

The startProgram method displays a menu to the user and asks for input.

If you choose to login, you will be asked for your client number and password. else quit the program.

If the client number and password are correct,

you will be logged in and get access to the costumer dashboard menu.

In this menu you can deposit money, withdraw money, save all data in files,

load data from files and quit the program.

If the user enters D, W, M, S or L, then the program calls the appropriate method on the BankUI object.

If the user enters **Q**, then the program exits.

If the user enters something else, then it displays an error message and asks for input again.

The **deposit** method asks for an account number and an amount to deposit. It then calls the deposit method on that account with that amount as a parameter. If there is no such account, it displays an error message. Otherwise it displays a success message and prints out all accounts in alphabetical order by name of owner.

The **withdraw** method asks for an account number and an amount to withdraw. It then calls the withdraw method on that account with that amount as a parameter. If there is no such account or if there is not enough money in that account to withdraw that amount, it displays an error message. Otherwise it displays a success message and prints out all accounts in alphabetical order by name of owner.

The **monthEnd** method calls monthEnd on all accounts in alphabetical order by name of owner and prints out all accounts in alphabetical order by name of owner after doing so.

The **saveToFile** method asks for a filename to save to and saves all accounts to that file in alphabetical order by name of owner using Java serialization (ObjectOutputStream). If there is any problem saving to file (e.g., file not found), it displays an error message instead of saving anything to file. Otherwise it displays a success message after saving everything to file successfully.

The **loadFromFile** method asks for a filename to load from and loads all accounts from that file using Java deserialization (ObjectInputStream). If there is any problem loading from file (e.g., file not found), it displays an error message instead of loading anything from file or adding anything to its list of accounts (i.e., its list of accounts should be empty after this). Otherwise it displays a success message after loading everything from file successfully and adds everything loaded from file into its list of accounts (i.e., its list of accounts should contain everything loaded from file after this).

If there are more than 3 withdrawals from this account

this month, a fee of 1 dollar they

use at this bank will also be withdrawn from this account

when withdrawing money from it.

If you choose to save all data in files, all data about accounts

and transactions at this bank will be saved

in files on your computer .

The data about accounts are saved in two different files: one file with

data about saving accounts and one file with data about checking accounts.

The data about transactions are saved in one file with all transactions

at this bank . A message saying that

all data were successfully saved appears after saving all data in files

on your computer .

Then you return to the costumer dashboard menu where you can do other

things like depositing or withdrawing money or quitting the program or

whatever else people do at banks .

If you choose to load data from files, all data from file will be loaded.