**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский университет ИТМО»**

Факультет информационных технологий и программирования

Лабораторная работа № 1

*Тема Банки*

**Выполнил студент группы M32051**

Писарева Юлия Игоревна

****

**Подпись:**

**Проверил:**

Чикишев Константин Максимович

Санкт-Петербург

2022

**Текст задания**

Есть несколько Банков, которые предоставляют финансовые услуги по операциям с деньгами.

В банке есть Счета и Клиенты. У клиента есть имя, фамилия, адрес и номер паспорта (имя и фамилия обязательны, остальное – опционально).

Счета и проценты Счета бывают трёх видов: Дебетовый счет, Депозит и Кредитный счет. Каждый счет принадлежит какому-то клиенту.

Дебетовый счет – обычный счет с фиксированным процентом на остаток. Деньги можно снимать в любой момент, в минус уходить нельзя. Комиссий нет.

Депозитный счет – счет, с которого нельзя снимать и переводить деньги до тех пор, пока не закончится его срок (пополнять можно). Процент на остаток зависит от изначальной суммы, например, если открываем депозит до 50 000 р. - 3%, если от 50 000 р. до 100 000 р. - 3.5%, больше 100 000 р. - 4%. Комиссий нет. Проценты должны задаваться для каждого банка свои.

Кредитный счет – имеет кредитный лимит, в рамках которого можно уходить в минус (в плюс тоже можно). Процента на остаток нет. Есть фиксированная комиссия за использование, если клиент в минусе.

Комиссии Периодически банки проводят операции по выплате процентов и вычету комиссии. Это значит, что нужен механизм проматывания времени, чтобы посмотреть, что будет через день/месяц/год и т.п.

Процент на остаток начисляется ежедневно от текущей суммы в этот день, но выплачивается раз в месяц (и для дебетовой карты и для депозита). Например, 3.65% годовых. Значит в день: 3.65% / 365 дней = 0.01%. У клиента сегодня 100 000 р. на счету - запомнили, что у него уже 10 р. Завтра ему пришла ЗП и стало 200 000 р. За этот день ему добавили ещё 20 р. На следующий день он купил себе новый ПК и у него осталось 50 000 р. - добавили 5 р. Таким образом, к концу месяца складываем все, что запоминали. Допустим, вышло 300 р. - эта сумма добавляется к счету или депозиту в текущем месяце.

Разные банки предлагают разные условия. В каждом банке известны величины процентов и комиссий.

Центральный банк Регистрацией всех банков, а также взаимодействием между банками занимается центральный банк. Он должен управлять банками (предоставлять возможность создать банк) и предоставлять необходимый функционал, чтобы банки могли взаимодействовать с другими банками (например, можно реализовать переводы между банками через него). Он также занимается уведомлением других банков о том, что нужно начислять остаток или комиссию - для этого механизма не требуется создавать таймеры и завязываться на реальное время.

Операции и транзакции Каждый счет должен предоставлять механизм снятия, пополнения и перевода денег (то есть счетам нужны некоторые идентификаторы).

Еще обязательный механизм, который должны иметь банки - отмена транзакций. Если вдруг выяснится, что транзакция была совершена злоумышленником, то такая транзакция должна быть отменена. Отмена транзакции подразумевает возвращение банком суммы обратно. Транзакция не может быть повторно отменена.

Создание клиента и счета Клиент должен создаваться по шагам. Сначала он указывает имя и фамилию (обязательно), затем адрес (можно пропустить и не указывать), затем паспортные данные (можно пропустить и не указывать).

Если при создании счета у клиента не указаны адрес или номер паспорта, мы объявляем такой счет (любого типа) сомнительным, и запрещаем операции снятия и перевода выше определенной суммы (у каждого банка своё значение). Если в дальнейшем клиент указывает всю необходимую информацию о себе - счет перестает быть сомнительным и может использоваться без ограничений.

Обновление условий счетов Для банков требуется реализовать методы изменений процентов и лимитов не перевод. Также требуется реализовать возможность пользователям подписываться на информацию о таких изменениях - банк должен предоставлять возможность клиенту подписаться на уведомления. Стоит продумать расширяемую систему, в которой могут появится разные способы получения нотификаций клиентом (да, да, это референс на тот самый сайт). Например, когда происходит изменение лимита для кредитных карт – все пользователи, которые подписались и имеют кредитные карты, должны получить уведомление.

Консольный интерфейс работы Для взаимодействия с банком требуется реализовать консольный интерфейс, который будет взаимодействовать с логикой приложения, отправлять и получать данные, отображать нужную информацию и предоставлять интерфейс для ввода информации пользователем.

Дополнения

На усмотрение студента можно ввести свои дополнительные идентификаторы для пользователей, банков etc.

На усмотрение студента можно пользователю добавить номер телефона или другие характеристики, если есть понимание зачем это нужно.

QnA Q: Нужно ли предоставлять механизм отписки от информации об изменениях в условии счетов A: Не обговорено, значит на ваше усмотрение (это вообще не критичный момент судя по условию лабы)

Q: Транзакциями считаются все действия со счётом, или только переводы между счетами. Если 1, то как-то странно поддерживать отмену операции снятия, а то после отмены деньги удвоятся: они будут и у злоумышленника на руках и на счету. Или просто на это забить A: Все операции со счетами - транзакции.

Q: Фиксированная комиссия за использование кредитного счёта, когда тот в минусе измеряется в % или рублях, и когда её начислять: после выполнения транзакции, или до. И нужно ли при отмене транзакции убирать и начисленную за неё комиссию. A: Фиксированная комиссия означает, что это фиксированная сумма, а не процент. Да, при отмене транзакции стоит учитывать то, что могла быть также комиссия.

Q: Если транзакция подразумевает возвращение суммы обратно - но при этом эта же сумма была переведена на несколько счетов (пример перевод денег со счета 1 на счёт 2, со счёта 2 на счёт 3) Что происходит если клиент 1 отменяет транзакцию? Подразумевается ли что деньги по цепочке снимаются со счёта 3? (на счету 2 их уже физически нет) Либо у нас банк мошеннический и деньги "отмываются" и возмещаются клиенту 1 с уводом счёта 2 в минус A: Банк не мошеннический, просто упрощённая система. Транзакции не связываются между собой. Так что да, можно считать, что может уйти в минус.

Иными словами: переписать лабораторную 4 из курса по ООП на Java

**Решение с комментариями**

package com.company.lab1.banks.dataTime;

import java.util.Calendar;

import java.util.Date;

public class DataTime {

public int NOWDAY;

public int NOWMONTH;

public int NOWYEAR;

Calendar calendar = Calendar.getInstance();

public DataTime() {

Date date = new Date();

calendar.setTime(date);

NOWDAY = calendar.get(Calendar.DAY\_OF\_MONTH);;

NOWMONTH = calendar.get(Calendar.MONTH) + 1;

NOWYEAR = calendar.get(Calendar.YEAR);

}

public boolean lessData(DataTime dataTime) {

if (NOWYEAR > dataTime.NOWYEAR) return false;

else if (NOWYEAR < dataTime.NOWYEAR) return true;

else if (NOWMONTH > dataTime.NOWMONTH) return false;

else if (NOWMONTH < dataTime.NOWMONTH) return true;

else return NOWDAY <= dataTime.NOWDAY;

}

}

package com.company.lab1.banks.entities.banksAccounts;

import com.company.lab1.banks.dataTime.DataTime;

import com.company.lab1.banks.entities.Bank;

import com.company.lab1.banks.entities.client.Client;

public interface BankAccount {

public void appointСommission(double commission); // комиссия

public void appointPercentages(double proc); // проценты

public void addingInterestToTheAmount();

public double getTheAmountOnTheAccount(); // сумма

public double cashWithdrawal(double sum, DataTime dateTime);

public double topUpYourAccount(double sum, DataTime dateTime);

public void transferOfMoney(double sum, Client person, DataTime dateTime);

public void deleteLastTransaction();

public void setMyBank(Bank bank);

public double getPercentages();

public double getCommission();

}

package com.company.lab1.banks.entities.banksAccounts;

import com.company.lab1.banks.dataTime.DataTime;

import com.company.lab1.banks.entities.Bank;

import com.company.lab1.banks.entities.client.Client;

import com.company.lab1.banks.services.queue.QueueOfResponsibilities;

import org.javatuples.Triplet;

public class Credit implements BankAccount{

/\* имеет кредитный лимит, в рамках которого можно уходить в минус (в плюс тоже можно).

Процента на остаток нет. Есть фиксированная комиссия за использование, если клиент в минусе.\*/

public Credit(double commission) {

Commission = commission;

Amount = 0;

}

private Bank MyBank;

private double Commission;

private double Amount;

private QueueOfResponsibilities Queue = new QueueOfResponsibilities();

private Triplet<String, Double, DataTime> LastTransaction;

private void setAmount(double amount) { Amount = amount; }

private void setLastTransaction(Triplet<String, Double, DataTime> transaction) { LastTransaction = transaction; }

private double getAmount() { return Amount; }

private Triplet<String, Double, DataTime> getLastTransaction() { return LastTransaction; }

public void setMyBank(Bank bank) { MyBank = bank; }

public double getPercentages() { return 0; }

public double getCommission() { return Commission; }

public double getTheAmountOnTheAccount() { return Amount; }

public void appointСommission(double commission) {

if (LastTransaction==null) return;

DataTime dataTime = new DataTime();

if (Amount < 0 && MyBank.getMyCentralBank().dayPassed(LastTransaction.getValue2(), dataTime)) Amount -= Commission;

}

public void appointPercentages(double proc) { }

public void addingInterestToTheAmount() { }

public double cashWithdrawal(double sum, DataTime dateTime) {

Amount -= sum;

var t = new Triplet<String, Double, DataTime>("-", sum, dateTime);

Queue.bankingOperation(this);

LastTransaction = t;

return sum;

}

public double topUpYourAccount(double sum, DataTime dateTime) {

Amount += sum;

var t = new Triplet<String, Double, DataTime>("-", sum, dateTime);

LastTransaction = t;

return Amount;

}

public void transferOfMoney(double sum, Client person, DataTime dateTime) {

Amount -= sum;

person.setMoney(sum, dateTime);

var t = new Triplet<String, Double, DataTime>("-", sum, dateTime);

LastTransaction = t;

}

public void deleteLastTransaction() { // у мошенников деньги мы уже не заберем

if (LastTransaction.getValue0() == "+") {

cashWithdrawal(LastTransaction.getValue1(), LastTransaction.getValue2());

}

else {

topUpYourAccount(LastTransaction.getValue1(), LastTransaction.getValue2());

}

LastTransaction = null;

}

}

package com.company.lab1.banks.entities.banksAccounts;

import com.company.lab1.banks.dataTime.DataTime;

import com.company.lab1.banks.entities.Bank;

import com.company.lab1.banks.entities.client.Client;

import com.company.lab1.banks.services.queue.QueueOfResponsibilities;

import org.javatuples.Triplet;

public class Debit implements BankAccount {

/\* обычный счет с фиксированным процентом на остаток.

Деньги можно снимать в любой момент, в минус уходить нельзя.

Комиссий нет \*/

public Debit(double sum, double proc) {

var Time = new DataTime();

Amount = sum;

Percentages = proc;

Sum = Amount \* Percentages;

LastTransaction = new Triplet<String, Double, DataTime>(" ", 0., Time);

}

public Debit(double sum) {

var Time = new DataTime();

Amount = sum;

Sum = Amount \* Percentages;

LastTransaction = new Triplet<String, Double, DataTime>(" ", 0., Time);

}

private Bank MyBank;

private QueueOfResponsibilities Queue = new QueueOfResponsibilities();

private double Amount;

private double Sum;

private double Percentages;

private Triplet<String, Double, DataTime> LastTransaction;

private void setSum(double sum) { Sum = sum; }

private void setLastTransaction(Triplet<String, Double, DataTime> lastTransaction) { LastTransaction = lastTransaction; }

private void setAmount(double amount) { Amount = amount; }

private double getSum() { return Sum; }

private Triplet<String, Double, DataTime> getLastTransaction() { return LastTransaction; }

private double getAmount() { return Amount; }

public void setPercentage(double proc) { Percentages = proc; }

public void setMyBank(Bank bank) { MyBank = bank; }

public double getTheAmountOnTheAccount() { return Amount; }

public double getPercentages() { return Percentages; }

public double getCommission() { return 0; }

public void appointСommission(double commission) { }

public void appointPercentages(double proc) {

DataTime dataTime = new DataTime();

if (MyBank.getMyCentralBank().dayPassed(LastTransaction.getValue2(), dataTime)) Sum += Amount \* Percentages;

}

public void addingInterestToTheAmount() {

DataTime dataTime = new DataTime();

if (MyBank.getMyCentralBank().monthPassed(LastTransaction.getValue2(), dataTime)) Amount += Sum;

}

public double cashWithdrawal(double sum, DataTime dateTime) {

if (!(Amount > sum)) return 0; // столько мы можем снять (ложный код возврата)

Amount -= sum;

var t = new Triplet<String, Double, DataTime>("+", sum, dateTime);

Queue.bankingOperation(this);

LastTransaction = t;

return sum;

}

public double topUpYourAccount(double sum, DataTime dateTime) {

Amount += sum;

var t = new Triplet<String, Double, DataTime>("-", sum, dateTime);

Queue.bankingOperation(this);

LastTransaction = t;

return Amount;

}

public void transferOfMoney(double sum, Client person, DataTime dateTime) {

if (!(sum < Amount) || person == null) return;

Amount -= sum;

person.setMoney(sum, dateTime);

var t = new Triplet<String, Double, DataTime>("-", sum, dateTime);

Queue.bankingOperation(this);

LastTransaction = t;

}

public void deleteLastTransaction() { // у мошенников деньги мы уже не заберем

if (LastTransaction.getValue0() == "+") {

cashWithdrawal(LastTransaction.getValue1(), LastTransaction.getValue2());

}

else {

topUpYourAccount(LastTransaction.getValue1(), LastTransaction.getValue2());

}

LastTransaction = null;

}

}

package com.company.lab1.banks.entities.banksAccounts;

import com.company.lab1.banks.dataTime.DataTime;

import com.company.lab1.banks.entities.Bank;

import com.company.lab1.banks.entities.client.Client;

import com.company.lab1.banks.entities.methods.percentage.MethodPercentageChange;

import com.company.lab1.banks.entities.methods.percentage.PercentageChange;

import com.company.lab1.banks.services.queue.QueueOfResponsibilities;

import org.javatuples.Triplet;

public class Deposit implements BankAccount {

/\* счет, с которого нельзя снимать и переводить деньги до тех пор, пока не закончится его срок (пополнять можно).

Процент на остаток зависит от изначальной суммы, например, если открываем депозит до 50 000 р. - 3%,

если от 50 000 р.до 100 000 р.- 3.5%, больше 100 000 р.- 4%.

Комиссий нет.

Проценты должны задаваться для каждого банка свои.\*/

public Deposit(double sum, DataTime date, MethodPercentageChange percentageChange) {

Amount = sum;

NowPercentages = getPercentages(Amount, percentageChange);

Date = date;

Sum = Amount \* getPercentages(Amount, percentageChange);

PercentageChange = percentageChange;

}

private Bank MyBank;

private QueueOfResponsibilities Queue = new QueueOfResponsibilities();

private double Amount;

private double Sum;

private Triplet<String, Double, DataTime> LastTransaction;

private double NowPercentages;

private MethodPercentageChange PercentageChange;

private void setAmount(double amount) { Amount=amount; }

private void setSum(double sum) { Sum=sum; }

private void setLastTransaction(Triplet<String, Double, DataTime> lastTransaction) { LastTransaction=lastTransaction; }

private double getSum() { return Sum; }

private Triplet<String, Double, DataTime> getLastTransaction() { return LastTransaction; }

private double getNowPercentages() { return NowPercentages; }

public DataTime Date;

public DataTime getDate() { return Date; }

private double getPercentages(double sum, MethodPercentageChange method) { return method.getPercentage(sum); }

public void setMyBank(Bank bank) { MyBank = bank; }

public void setPercentageMethod(MethodPercentageChange methodPercentageChange) { PercentageChange = methodPercentageChange; }

public double getPercentages() { return 0; }

public double getCommission() { return 0; }

public void appointСommission(double commission) { }

public double getTheAmountOnTheAccount() { return Amount; }

public void appointPercentages(double proc) {

if (LastTransaction==null) return;

DataTime dataTime = new DataTime();

if (MyBank.getMyCentralBank().dayPassed(LastTransaction.getValue2(), dataTime)) Sum += Amount \* getPercentages(Amount, PercentageChange);

}

public void addingInterestToTheAmount() { // когда позволит ценральный банк

if (LastTransaction==null) return;

DataTime dataTime = new DataTime();

if (MyBank.getMyCentralBank().monthPassed(LastTransaction.getValue2(), dataTime)) Amount += Sum;

}

public double cashWithdrawal(double sum, DataTime dateTime) {

if (dateTime.lessData(Date) || !(Amount > sum)) return 0;

Amount -= sum;

var t = new Triplet<String, Double, DataTime>("-", sum, dateTime);

Queue.bankingOperation(this);

LastTransaction = t;

return sum;

}

public double topUpYourAccount(double sum, DataTime dateTime) {

Amount += sum;

var t = new Triplet<String, Double, DataTime>("-", sum, dateTime);

Queue.bankingOperation(this);

LastTransaction = t;

return Amount;

}

public void transferOfMoney(double sum, Client person, DataTime dateTime) {

if (dateTime.lessData(Date)) return;

if (!(sum < Amount) || person == null) return;

Amount -= sum;

person.setMoney(sum, dateTime);

var t = new Triplet<String, Double, DataTime>("-", sum, dateTime);

Queue.bankingOperation(this);

LastTransaction = t;

}

public void deleteLastTransaction() { // у мошенников деньги мы уже не заберем

if (LastTransaction.getValue0() == "+") {

cashWithdrawal(LastTransaction.getValue1(), LastTransaction.getValue2());

}

else {

topUpYourAccount(LastTransaction.getValue1(), LastTransaction.getValue2());

}

LastTransaction = null;

}

}

package com.company.lab1.banks.entities.client;

import com.company.lab1.banks.dataTime.DataTime;

import com.company.lab1.banks.entities.banksAccounts.BankAccount;

import java.util.Objects;

public class Client {

private final int MAXPASSPORT = 999999999;

private final int MINPASSPORT = 1000000000;

private String fakeAdres = " ";

private int fakePassport = 0;

public Client(String name, int id, String address, int passport) {

Name = name;

Address = address;

Passport = passport;

ID = id;

BankAccount = null;

}

public Client(String name, int id) {

Name = name;

ID = id;

BankAccount = null;

}

public Client(String name, int id, int passport) {

Name = name;

Passport = passport;

ID = id;

BankAccount = null;

}

public Client(String name, int id, String address) {

Name = name;

Address = address;

ID = id;

BankAccount = null;

}

private String Name;

private String getName() { return Name; }

public String Address;

public int Passport;

public int Phone;

public int ID;

public BankAccount BankAccount;

public void setPassport(int passport) { Passport=passport; }

public void setPhone(int phone) { Phone=phone; }

public void setAddress(String address) { Address=address; }

public void setBankAccount(BankAccount bankAccount) { BankAccount=bankAccount; }

public String getAddress() { return Address; }

public int getNumberPassport() { return Passport; }

public BankAccount getBankAccount() { return BankAccount; }

public int getPhone() { return Phone; }

public int getID() { return ID; }

public boolean getAdress() { return !Objects.equals(Address, fakeAdres); }

public boolean getPassport() { return Passport != fakePassport; }

public BankAccount getAccount() { return BankAccount; }

public void setMoney(double sum, DataTime dateTime) { BankAccount.topUpYourAccount(sum, dateTime); }

}

package com.company.lab1.banks.entities.methods.percentage;

public interface MethodPercentageChange {

public double getPercentage(double sum);

}

package com.company.lab1.banks.entities.methods.percentage;

import org.javatuples.Triplet;

import java.util.ArrayList;

public class PercentageChange implements MethodPercentageChange {

public ArrayList<Triplet<Integer, Integer, Double>> Table = new ArrayList<Triplet<Integer, Integer, Double>>();

public PercentageChange() { Table = new ArrayList<Triplet<Integer, Integer, Double>>(); }

public ArrayList<Triplet<Integer, Integer, Double>> getTable() { return Table; }

public void addСondition(int start, int finish, double proc) {

Triplet<Integer, Integer, Double> tuple = new Triplet<Integer, Integer, Double>(start, finish, proc);

for (int i=0; i < Table.size(); i++) {

if (Table.get(i).getValue0() <= start || Table.get(i).getValue0() >= finish) continue;

Triplet<Integer, Integer, Double> newTuple = new Triplet<Integer, Integer, Double>(finish, Table.get(i).getValue1(), Table.get(i).getValue2());

Table.remove(Table.get(i));

Table.add(newTuple);

}

Table.add(tuple);

}

public double getPercentage(double sum) {

for (Triplet<Integer, Integer, Double> objects : Table) {

if (objects.getValue0() < sum && objects.getValue1() > sum) return sum \* objects.getValue2();

}

return sum;

}

}

package com.company.lab1.banks.entities.methods;

public interface MethodTransferLimit {

public double getMaxSum();

}

package com.company.lab1.banks.entities.methods;

public class TransferLimit implements MethodTransferLimit {

public TransferLimit(double sum){

Sum = sum;

}

private double Sum;

private double getSum() {

return Sum;

}

public double getMaxSum(){

return Sum;

}

}

package com.company.lab1.banks.entities;

import com.company.lab1.banks.dataTime.DataTime;

import com.company.lab1.banks.entities.banksAccounts.BankAccount;

import com.company.lab1.banks.entities.client.Client;

import com.company.lab1.banks.entities.methods.TransferLimit;

import com.company.lab1.banks.entities.methods.percentage.MethodPercentageChange;

import com.company.lab1.banks.services.CentralBank;

import org.javatuples.Pair;

import java.util.ArrayList;

public class Bank implements Banks {

public Bank(int id)

{

ID = id;

Clients = new ArrayList<Pair<Client, BankAccount>>();

ClientID = 0;

TransferLimit = new TransferLimit(Double.MAX\_VALUE);

}

private CentralBank MyCentralBank;

private int ID;

private int ClientID;

private TransferLimit TransferLimit;

private TransferLimit DoubtfulLimit;

private MethodPercentageChange PercentageChange;

private ArrayList<Pair<Client, BankAccount>> Clients;

private void setTransferLimit(TransferLimit transferLimit) { TransferLimit = transferLimit; }

private void setPercentageChange(MethodPercentageChange method) { PercentageChange = method; }

private void setClientID(int id) { ClientID = id;}

private Client setClient(String name, String address, int passport) {

return new Client(name, ClientID, address, passport);

}

private int getClientID() { return ClientID; }

private ArrayList<Pair<Client, BankAccount>> getClients() { return Clients; }

public MethodPercentageChange getPercentageChange(){ return PercentageChange; }

public void setMethodPercentageChange(MethodPercentageChange percentageChange) { PercentageChange = percentageChange; }

public void setMethodTransferLimit(TransferLimit transferLimit) { TransferLimit = transferLimit; }

public void setDoubtfulLimit(TransferLimit transferLimit) { DoubtfulLimit = transferLimit; }

public void setMyCentralBank(CentralBank cb) { MyCentralBank = cb; }

public int getID() { return ID; }

public TransferLimit getDoubtfulLimit() { return DoubtfulLimit; }

public TransferLimit getTransferLimit() { return TransferLimit; }

public CentralBank getMyCentralBank() { return MyCentralBank; }

public Client addClient(String name, BankAccount account, String address, int passport) {

Pair<Client, BankAccount> client = new Pair<Client, BankAccount>(setClient(name, address, passport), account);

Clients.add(client);

ClientID++;

return client.getValue0();

}

public void addClient(Client person) {

var t = new Pair<Client, BankAccount>(person, person.getAccount());

Clients.add(t);

}

public double getMoney(Client person, double sum, DataTime dateTime) {

for (Pair<Client, BankAccount> client : Clients) {

if (client.getValue0().getID() == person.getID()) {

if (client.getValue0().getAdress() && client.getValue0().getPassport()) return client.getValue1().cashWithdrawal(sum, dateTime);

return client.getValue1().cashWithdrawal(sum < DoubtfulLimit.getMaxSum() ? sum : DoubtfulLimit.getMaxSum(), dateTime);

}

}

return 0;

}

}

package com.company.lab1.banks.entities;

import com.company.lab1.banks.entities.methods.TransferLimit;

import com.company.lab1.banks.entities.methods.percentage.MethodPercentageChange;

public interface Banks {

/\* В банке есть Счета и Клиенты.

Для банков требуется реализовать методы изменений процентов и лимитов нa перевод.

Также требуется реализовать возможность пользователям подписываться на информацию о таких изменениях -

банк должен предоставлять интерфейс для подписывания. Например, когда происходит изменение лимита для

кредитных карт - все пользователи, которые подписались и имеют кредитные карты, должны получить уведомление.\*/

public void setMethodPercentageChange(MethodPercentageChange percentageChange);

public void setMethodTransferLimit(TransferLimit transferLimit);

}

package com.company.lab1.banks.services.builder;

import com.company.lab1.banks.entities.banksAccounts.BankAccount;

import com.company.lab1.banks.entities.client.Client;

public class AddendumClientBuilder implements ClientBuilders{

private final int MAXPASSPORT = 999999999;

private final int MINPASSPORT = 1000000000;

private String fakeAdres = " ";

private int fakePassport = 0;

public AddendumClientBuilder(Client client) { Client = client; }

private Client Client;

public Client Build() { return Client; }

public void setName(String name, int id) { }

public void setAddress(String address) { Client.Address = address; }

public void setAccount(BankAccount account) { Client.BankAccount = account; }

public void setPhone(int phone) { Client.Phone = phone; }

public void setPassport(int passport) {

if (passport <= MAXPASSPORT && passport >= MINPASSPORT) Client.Passport = passport;

}

}

package com.company.lab1.banks.services.builder;

import com.company.lab1.banks.entities.banksAccounts.BankAccount;

import com.company.lab1.banks.entities.client.Client;

import com.company.lab1.banks.tools.Exceptions;

public class ClientBuilder implements ClientBuilders {

private final int MAXPASSPORT = 999999999;

private final int MINPASSPORT = 1000000000;

private String fakeAdres = " ";

private int fakePassport = 0;

private Client Client;

public Client Build() {

if (Client != null) return Client;

throw new Exceptions("the client does not exist");

}

public void setName(String name, int id) { Client = new Client(name, id, fakeAdres, fakePassport); }

public void setPassport(int passport) { if (passport <= MAXPASSPORT && passport >= MINPASSPORT) Client.Passport = passport; }

public void setAddress(String address) { Client.Address = address; }

public void setAccount(BankAccount account) { Client.BankAccount = account; }

}

package com.company.lab1.banks.services.builder;

import com.company.lab1.banks.entities.banksAccounts.BankAccount;

import com.company.lab1.banks.entities.client.Client;

public interface ClientBuilders {

public Client Build();

public void setName(String name, int id);

public void setPassport(int passport);

public void setAddress(String address);

public void setAccount(BankAccount account);

}

package com.company.lab1.banks.services.factory;

import com.company.lab1.banks.entities.banksAccounts.BankAccount;

public interface Factory {

public void setAccount(BankAccount account);

public BankAccount createdBankAccount(int account);

}

package com.company.lab1.banks.services.factory;

import com.company.lab1.banks.dataTime.DataTime;

import com.company.lab1.banks.entities.Bank;

import com.company.lab1.banks.entities.banksAccounts.BankAccount;

import com.company.lab1.banks.entities.banksAccounts.Credit;

import com.company.lab1.banks.entities.banksAccounts.Debit;

import com.company.lab1.banks.entities.banksAccounts.Deposit;

import java.util.Scanner;

import static java.lang.System.in;

import static java.lang.System.out;

public class FactoryConsole implements Factory {

private BankAccount Account;

private double fakeComissions = 0;

private double fakeProcent = 0;

private Bank Bank;

public void setAccount(BankAccount account) { Account = account; }

public FactoryConsole(Bank bank) {

Bank = bank;

}

public BankAccount createdBankAccount(int account) {

var scanner = new Scanner(in);

double sum;

BankAccount bankAccount = null;

switch (account) {

case 1, 2 -> {

out.print("Какую сумму Вы кладете на счет? ");

sum = Double.parseDouble(scanner.nextLine());

switch (account) {

case 1 -> bankAccount = new Debit(sum, fakeProcent);

case 2 -> {

DataTime dataTime = new DataTime();

bankAccount = new Deposit(sum, dataTime, Bank.getPercentageChange());

}

}

}

case 3 -> {

sum = Double.MAX\_VALUE;

bankAccount = new Credit(fakeComissions);

}

}

return bankAccount;

}

}

package com.company.lab1.banks.services.queue;

import com.company.lab1.banks.entities.banksAccounts.BankAccount;

public interface Queue {

public void bankingOperation(BankAccount account);

}

package com.company.lab1.banks.services.queue;

import com.company.lab1.banks.entities.banksAccounts.BankAccount;

public class QueueOfResponsibilities implements Queue {

public void bankingOperation(BankAccount account) {

account.appointPercentages(account.getPercentages());

account.addingInterestToTheAmount();

account.appointСommission(account.getCommission());

}

}

package com.company.lab1.banks.services;

import com.company.lab1.banks.dataTime.DataTime;

import com.company.lab1.banks.entities.Bank;

import com.company.lab1.banks.entities.Banks;

import com.company.lab1.banks.entities.methods.TransferLimit;

import com.company.lab1.banks.entities.methods.percentage.MethodPercentageChange;

import java.util.ArrayList;

public class CentralBank {

/\* Регистрацией всех банков, а также взаимодействием между банками занимается центральный банк.

Он должен предоставлять все нужные другим банкам методы, методы добавления нового банка.

Он также занимается уведомлением других банков о том, что нужно начислять остаток или комиссию -

для этого механизма НЕ требуется создавать таймеры и завязываться на реальное время. \*/

public CentralBank()

{

Banks = new ArrayList<Bank>();

id = 0;

}

private int id;

private ArrayList<Bank> Banks;

private ArrayList<Bank> getBank() { return Banks; }

public void addBank(MethodPercentageChange percentageChange, TransferLimit transferLimit) {

var bank = new Bank(id);

id++;

provideTheMethodToTheBank(bank, percentageChange, transferLimit);

Banks.add(bank);

bank.setMyCentralBank(this);

}

public void addBank(Bank bank){ Banks.add(bank); }

public ArrayList<Bank> getListBanks() { return Banks; }

public void provideTheMethodToTheBank(Banks bank, MethodPercentageChange percentageChange, TransferLimit transferLimit) {

bank.setMethodPercentageChange(percentageChange);

bank.setMethodTransferLimit(transferLimit);

}

public boolean monthPassed(DataTime lastTime, DataTime nowTime) {

return (nowTime.NOWMONTH - lastTime.NOWMONTH == 1 && nowTime.NOWYEAR == lastTime.NOWYEAR) || (nowTime.NOWYEAR == lastTime.NOWYEAR + 1 && nowTime.NOWMONTH == 1 && lastTime.NOWMONTH == 12);

}

public boolean dayPassed(DataTime lastTime, DataTime nowTime) {

return nowTime.NOWYEAR == lastTime.NOWYEAR && nowTime.NOWMONTH == lastTime.NOWMONTH &&

nowTime.NOWDAY == lastTime.NOWDAY + 1;

}

}

package com.company.lab1.banks.services;

import com.company.lab1.banks.dataTime.DataTime;

import com.company.lab1.banks.entities.Bank;

import com.company.lab1.banks.entities.banksAccounts.BankAccount;

import com.company.lab1.banks.entities.client.Client;

import com.company.lab1.banks.entities.methods.TransferLimit;

import com.company.lab1.banks.entities.methods.percentage.MethodPercentageChange;

import com.company.lab1.banks.entities.methods.percentage.PercentageChange;

import com.company.lab1.banks.services.factory.FactoryConsole;

import java.util.Scanner;

import static java.lang.System.in;

import static java.lang.System.out;

public class ConsoleService implements ConsoleServices {

/\* Для взаимодействия с банком требуется реализовать консольный интерфейс,

который будет взаимодействовать с логикой приложения, отправлять и получать данные,

отображать нужную информацию и предоставлять интерфейс для ввода информации пользователем. \*/

private final int MAXPASSPORT = 999999999;

private final int MINPASSPORT = 1000000000;

private void makeBank(TransferLimit transferLimit, MethodPercentageChange percentageChange) {

var cb = new CentralBank();

cb.addBank(percentageChange, transferLimit);

}

private void makeBank(Bank bank) {

var cb = new CentralBank();

cb.addBank(bank);

bank.setMyCentralBank(cb);

}

public BankAccount enterData() {

Scanner scanner = new Scanner(in);

String address = null;

int passport = 0;

double sum = 0;

BankAccount bankAccount = null;

out.print("Введите Ваше полное имя: ");

String name = scanner.nextLine();

out.print("Хотите ли ввести свои данные о паспорте и месте жительства сейчас? (да или нет)");

String ans = scanner.nextLine();

if (ans == "да") {

out.print("Введите номер и сорию паспорта без пробела: ");

passport = Integer.parseInt(scanner.nextLine());

if (passport > MAXPASSPORT || passport < MINPASSPORT) out.print("Некорректно введен парспорт. Зайдите в ближайший банк для исправления.");

out.print("Введите свой адрес: ");

address = scanner.nextLine();

}

out.print("Каким счётом Вы хотели бы воспользоваться? (1-Дебетовый счет, 2-Депозит, 3-Кредитный счет) ");

int account = Integer.parseInt(scanner.nextLine());

var bank = new Bank(0);

makeBank(bank);

MethodPercentageChange methodPercentageChange = new PercentageChange();

bank.setMethodPercentageChange(methodPercentageChange);

var transferLimit = new TransferLimit(sum / 2);

makeBank(transferLimit, methodPercentageChange);

var myFactory = new FactoryConsole(bank);

bankAccount = myFactory.createdBankAccount(account);

out.print("Ваш банковский аккаунт создан. Спасибо за визит. ");

var person = new Client(name, 0, address, passport);

bankAccount.setMyBank(bank);

bank.addClient(name, bankAccount, address, passport);

return bankAccount;

}

public void seeMuchAccounts(BankAccount account) {

out.print("На вашем счёте: ");

out.println(account.getTheAmountOnTheAccount());

}

public void cashWithdrawal(BankAccount account, double sum) {

DataTime dataTime = new DataTime();

double s = account.cashWithdrawal(sum, dataTime);

if (s == 0) { out.print("Деньги снять не удалось. "); }

else {

out.print("На Вашем счёте ");

out.print(sum - s);

}

}

public void topUpYourAccount(BankAccount account, double sum) {

DataTime dataTime = new DataTime();

double s = account.topUpYourAccount(sum, dataTime);

out.print("На Вашем счёте ");

out.print(s);

}

}

package com.company.lab1.banks.services;

import com.company.lab1.banks.entities.banksAccounts.BankAccount;

public interface ConsoleServices {

public BankAccount enterData();

}

package com.company.lab1.banks.tools;

public class Exceptions extends RuntimeException {

private Integer errorCode;

public Exceptions(String message) {

super(message);

}

public Exceptions(String message, Throwable cause) {

super(message, cause);

}

public Integer getErrorCode() {

return errorCode;

}

}

import com.company.lab1.banks.entities.banksAccounts.BankAccount;

import com.company.lab1.banks.services.ConsoleService;

public class Main {

public static void main(String[] args) {

ConsoleService service = new ConsoleService();

BankAccount account = service.enterData();

service.seeMuchAccounts(account);

service.cashWithdrawal(account, 100);

service.topUpYourAccount(account, 1000);

}

}

package com.company.lab1.banksTest;

import com.company.lab1.banks.dataTime.DataTime;

import com.company.lab1.banks.entities.Bank;

import com.company.lab1.banks.entities.banksAccounts.BankAccount;

import com.company.lab1.banks.entities.banksAccounts.Debit;

import com.company.lab1.banks.entities.client.Client;

import com.company.lab1.banks.entities.methods.TransferLimit;

import com.company.lab1.banks.entities.methods.percentage.MethodPercentageChange;

import com.company.lab1.banks.entities.methods.percentage.PercentageChange;

import com.company.lab1.banks.services.CentralBank;

import com.company.lab1.banks.services.builder.ClientBuilder;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestInstance;

import static org.junit.jupiter.api.Assertions.assertEquals;

import static org.junit.jupiter.api.Assertions.assertTrue;

@TestInstance(TestInstance.Lifecycle.PER\_CLASS)

public class BanksTest {

private CentralBank CentralBank;

@BeforeAll

public void setup() { CentralBank = new CentralBank(); }

@Test

public void addBankToCentralBankList\_CentralBankListContainsBank() {

var bank = new Bank(0);

CentralBank.addBank(bank);

assertTrue(CentralBank.getListBanks().contains(bank));

}

@Test

public void theMoneyWasReturnedToTheClientAfterTheTransactionWasCanceled() {

double sum1 = 100000.5;

double sum2 = 100000.3;

var ClientBuilder = new ClientBuilder();

ClientBuilder.setName("Ксения Павловна", 1613923365);

BankAccount bankAccount1 = new Debit(sum1);

var transferLimit = new TransferLimit(sum1 / 2);

var bank = new Bank(0);

ClientBuilder.setAccount(bankAccount1);

ClientBuilder.setAddress("ул.Михалкова 14");

ClientBuilder.setPassport(1613923365);

Client Ksenia = ClientBuilder.Build();

MethodPercentageChange methodPercentageChange = new PercentageChange();

bank.setMethodPercentageChange(methodPercentageChange);

bank.setMethodTransferLimit(transferLimit);

var ClientBuilder2 = new ClientBuilder();

ClientBuilder2.setName("Лидия Михайловна", 1414778398);

BankAccount bankAccount2 = new Debit(sum2);

ClientBuilder2.setAccount(bankAccount2);

ClientBuilder2.setAddress("ул.Чуйкова 5");

ClientBuilder2.setPassport(1414778398);

Client Lida = ClientBuilder.Build();

bankAccount1.setMyBank(bank);

bankAccount2.setMyBank(bank);

CentralBank.addBank(bank);

bank.setMyCentralBank(CentralBank);

assertEquals(sum1, bankAccount1.getTheAmountOnTheAccount());

bankAccount1.transferOfMoney((sum1 / 4), Lida, new DataTime());

}

}