**Министерство науки и высшего образования Российской Федерации**

федеральное государственное автономное образовательное учреждение высшего образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**Отчет**

по лабораторной работе №1

по дисциплине «**Технологии программирования**»

Автор: Юрпалов Сергей Николаевич

Факультет: ИТиП

Группа: М32051



Санкт-Петербург 2022

Описание:

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.

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.

Исходный код:

Добавлен файл src/com/itmo/Main.java:

package com.itmo;

import com.itmo.builders.CustomerBuilder;

import com.itmo.console\_interface.ConsoleInterface;

import com.itmo.interfaces.AbstractBank;

import com.itmo.interfaces.AbstractCentralBank;

import com.itmo.models.Bank;

import com.itmo.models.CentralBank;

import com.itmo.tools.BanksException;

public class Main {

public static void main(String[] args) throws BanksException {

AbstractCentralBank centralBank = new CentralBank();

AbstractBank bank = new Bank(

100000,

10000,

5,

50000,

500);

centralBank.addBank(bank);

var customer= new CustomerBuilder().

setName("Sergey", "Yurpalov").

setAddress("123").

setPassport("123").

build();

new ConsoleInterface(centralBank, bank, customer).start();

}

}

Добавлен файл src/com/itmo/accounts/AbstractAccount.java:

package com.itmo.accounts;

import com.itmo.interfaces.AbstractCustomer;

import com.itmo.tools.BanksException;

import java.util.UUID;

public interface AbstractAccount {

UUID getId();

double getFunds();

AbstractCustomer getOwner();

AccountTypes getType();

double getTransactionLimitForRegisteredCustomer();

void setTransactionLimitForRegisteredCustomer(double limit);

double getTransactionLimitForUnregisteredCustomer();

void setTransactionLimitForUnregisteredCustomer(double limit);

void addFunds(double amount);

void withdrawFunds(double amount) throws BanksException;

void notifyOwner(String message);

void updateDay();

void updateMonth();

void updateTransactionLimit();

}

Добавлен файл src/com/itmo/accounts/AccountTypes.java:

package com.itmo.accounts;

public enum AccountTypes {

DEBIT\_ACCOUNT,

CREDIT\_ACCOUNT,

DEPOSIT\_ACCOUNT

}

Добавлен файл src/com/itmo/accounts/CreditAccount.java:

package com.itmo.accounts;

import java.util.UUID;

import com.itmo.interfaces.AbstractCustomer;

import com.itmo.tools.BanksException;

public class CreditAccount implements AbstractAccount {

private AbstractCustomer owner;

private double funds;

private final UUID id = UUID.randomUUID();

private double transactionLimit;

private double creditCommissionSum;

private double transactionLimitForRegisteredCustomer;

private double transactionLimitForUnregisteredCustomer;

private double fee;

private double creditAmount;

public CreditAccount(AbstractCustomer owner,

double transactionLimitForRegisteredCustomer,

double transactionLimitForUnregisteredCustomer,

double fee,

double creditAmount) {

this.owner = owner;

this.transactionLimitForRegisteredCustomer = transactionLimitForRegisteredCustomer;

this.transactionLimitForUnregisteredCustomer = transactionLimitForUnregisteredCustomer;

this.fee = fee;

this.creditAmount = creditAmount;

owner.getAccountIds().add(getId());

setTransactionLimit();

}

public AbstractCustomer getOwner() {

return owner;

}

private void setOwner(AbstractCustomer owner) {

this.owner = owner;

}

public double getTransactionLimitForRegisteredCustomer() {

return transactionLimitForRegisteredCustomer;

}

public void setTransactionLimitForRegisteredCustomer(double limit) {

this.transactionLimitForRegisteredCustomer = limit;

}

public double getTransactionLimitForUnregisteredCustomer() {

return transactionLimitForUnregisteredCustomer;

}

public void setTransactionLimitForUnregisteredCustomer(double limit) {

this.transactionLimitForUnregisteredCustomer = limit;

}

public double getFee() {

return fee;

}

public void setFee(double fee) {

this.fee = fee;

}

public double getCreditAmount() {

return creditAmount;

}

public void setCreditAmount(double creditAmount) {

this.creditAmount = creditAmount;

}

public double getFunds() {

return Math.round(funds);

}

public void setFunds(double funds) {

this.funds = funds;

}

public AccountTypes getType(){

return AccountTypes.CREDIT\_ACCOUNT;

}

public UUID getId() {

return id;

}

public void addFunds(double amount) {

funds += amount;

}

public void withdrawFunds(double amount) throws BanksException {

if (amount >= transactionLimit) {

throw new BanksException("Transaction limit of the account was exceeded");

}

if (funds + creditAmount < amount) {

throw new BanksException("Insufficient funds");

}

funds -= amount;

}

public void notifyOwner(String message) {

owner.update(message);

}

public void updateDay() {

if (funds < 0) {

creditCommissionSum += fee;

}

}

public void updateMonth() {

funds -= creditCommissionSum;

creditCommissionSum = 0;

}

public void updateTransactionLimit() {

setTransactionLimit();

}

public void changeCreditLimit(double creditLimit) throws BanksException {

if (creditLimit <= 0) {

throw new BanksException("Credit limit cannot be negative");

}

creditAmount = creditLimit;

}

public void changeCreditFee(double creditFee) throws BanksException {

if (creditFee <= 0) {

throw new BanksException("Credit fee cannot be negative");

}

fee = creditFee;

}

private void setTransactionLimit() {

if (owner.isVerified()){

transactionLimit = transactionLimitForRegisteredCustomer;

return;

}

transactionLimit = transactionLimitForUnregisteredCustomer;

}

}

Добавлен файл src/com/itmo/accounts/DebitAccount.java:

package com.itmo.accounts;

import java.util.UUID;

import com.itmo.interfaces.AbstractCustomer;

import com.itmo.tools.BanksException;

import com.itmo.tools.Config;

public class DebitAccount implements AbstractAccount {

private AbstractCustomer owner;

private double funds;

private final UUID id = UUID.randomUUID();

private double transactionLimit;

private double interestFunds;

private double transactionLimitForRegisteredCustomer;

private double transactionLimitForUnregisteredCustomer;

private double annualInterestOnBalance;

public DebitAccount(double transactionLimitForRegisteredCustomer,

double transactionLimitForUnregisteredCustomer,

double annualInterestOnBalance,

AbstractCustomer owner) {

this.transactionLimitForRegisteredCustomer = transactionLimitForRegisteredCustomer;

this.transactionLimitForUnregisteredCustomer = transactionLimitForUnregisteredCustomer;

this.annualInterestOnBalance = annualInterestOnBalance;

this.owner = owner;

owner.getAccountIds().add(getId());

setTransactionLimit();

}

public AbstractCustomer getOwner() {

return owner;

}

private void setOwner(AbstractCustomer owner) {

this.owner = owner;

}

public double getTransactionLimitForRegisteredCustomer() {

return transactionLimitForRegisteredCustomer;

}

public void setTransactionLimitForRegisteredCustomer(double limit) {

this.transactionLimitForRegisteredCustomer = limit;

}

public double getTransactionLimitForUnregisteredCustomer() {

return transactionLimitForUnregisteredCustomer;

}

public void setTransactionLimitForUnregisteredCustomer(double limit) {

this.transactionLimitForUnregisteredCustomer = limit;

}

public double getAnnualInterestOnBalance() {

return annualInterestOnBalance;

}

public void setAnnualInterestOnBalance(double annualInterestOnBalance) {

this.annualInterestOnBalance = annualInterestOnBalance;

}

public double getFunds() {

return Math.round(funds);

}

public void setFunds(double funds) {

this.funds = funds;

}

public AccountTypes getType(){

return AccountTypes.DEBIT\_ACCOUNT;

}

public UUID getId() {

return id;

}

public void addFunds(double amount) {

funds += amount;

}

public void withdrawFunds(double amount) throws BanksException {

if (funds <= 0) {

throw new BanksException("Insufficient funds");

}

if (amount >= transactionLimit) {

throw new BanksException("Transaction limit of the account was exceeded");

}

if (funds < amount) {

throw new BanksException("Insufficient funds");

}

funds -= amount;

}

public void calculateInterest() {

interestFunds += annualInterestOnBalance /

Config.PERCENT\_DIVIDER /

Config.DAYS\_IN\_YEAR \* funds;

}

public void notifyOwner(String message) {

owner.update(message);

}

public void updateDay() {

calculateInterest();

}

public void updateMonth() {

funds += interestFunds;

interestFunds = 0;

}

public void updateTransactionLimit() {

setTransactionLimit();

}

public void changeAnnualInterest(double interest) throws BanksException {

if (interest <= 0) {

throw new BanksException("Annual interest cannot be negative");

}

annualInterestOnBalance = interest;

}

private void setTransactionLimit() {

if (owner.isVerified()){

transactionLimit = transactionLimitForRegisteredCustomer;

return;

}

transactionLimit = transactionLimitForUnregisteredCustomer;

}

}

Добавлен файл src/com/itmo/accounts/DepositAccount.java:

package com.itmo.accounts;

import com.itmo.interfaces.AbstractCustomer;

import com.itmo.tools.BanksException;

import com.itmo.tools.Config;

import java.util.UUID;

public class DepositAccount implements AbstractAccount{

private AbstractCustomer owner;

private double funds;

private final UUID id = UUID.randomUUID();

private int daysToExpire;

private double interestFunds;

private double transactionLimit;

private double interestOnBalance;

private double transactionLimitForRegisteredCustomer;

private double transactionLimitForUnregisteredCustomer;

public DepositAccount(double transactionLimitForRegisteredCustomer,

double transactionLimitForUnregisteredCustomer,

double funds,

int daysToExpire,

AbstractCustomer owner) {

this.transactionLimitForRegisteredCustomer = transactionLimitForRegisteredCustomer;

this.transactionLimitForUnregisteredCustomer = transactionLimitForUnregisteredCustomer;

this.funds = funds;

this.daysToExpire = daysToExpire;

this.owner = owner;

owner.getAccountIds().add(getId());

setTransactionLimit();

setInterestByStartFunds(funds);

}

private void setDaysToExpire(int daysToExpire) {

this.daysToExpire = daysToExpire;

}

public AbstractCustomer getOwner() {

return owner;

}

private void setOwner(AbstractCustomer owner) {

this.owner = owner;

}

public double getTransactionLimitForRegisteredCustomer() {

return transactionLimitForRegisteredCustomer;

}

public void setTransactionLimitForRegisteredCustomer(double limit) {

this.transactionLimitForRegisteredCustomer = limit;

}

public double getTransactionLimitForUnregisteredCustomer() {

return transactionLimitForUnregisteredCustomer;

}

public void setTransactionLimitForUnregisteredCustomer(double limit) {

this.transactionLimitForUnregisteredCustomer = limit;

}

public double getFunds() {

return Math.round(funds);

}

public void setFunds(double funds) {

this.funds = funds;

}

public AccountTypes getType(){

return AccountTypes.DEPOSIT\_ACCOUNT;

}

public UUID getId() {

return id;

}

public void addFunds(double amount)

{

funds += amount;

}

public void withdrawFunds(double amount) throws BanksException {

if (funds <= 0) {

throw new BanksException("Insufficient funds");

}

if (daysToExpire > 0) {

throw new BanksException("Account is temporary blocked for withdrawal." + daysToExpire + "days left");

}

if (amount >= transactionLimit) {

throw new BanksException("While calling an operation, you have exceeded your transaction limit.");

}

if (funds < amount) {

throw new BanksException("Insufficient funds.");

}

funds -= amount;

}

public void calculateInterest()

{

interestFunds += interestOnBalance /

Config.PERCENT\_DIVIDER /

Config.DAYS\_IN\_YEAR \* funds;

}

public void updateDay()

{

calculateInterest();

if (daysToExpire > 0)

{

daysToExpire--;

}

}

public void updateMonth()

{

funds += interestFunds;

interestFunds = 0;

}

public void notifyOwner(String message)

{

owner.update(message);

}

public void updateTransactionLimit()

{

setTransactionLimit();

}

private void setTransactionLimit() {

if (owner.isVerified()){

transactionLimit = transactionLimitForRegisteredCustomer;

return;

}

transactionLimit = transactionLimitForUnregisteredCustomer;

}

private void setInterestByStartFunds(double funds)

{

if (funds <= 50000)

{

interestOnBalance = 3;

return;

}

else if (funds <= 100000)

{

interestOnBalance = 3.5;

return;

}

interestOnBalance = 4;

}

}

Добавлен файл src/com/itmo/banks\_tests/BanksTests.java:

package com.itmo.banks\_tests;

import com.itmo.accounts.CreditAccount;

import com.itmo.accounts.DebitAccount;

import com.itmo.accounts.DepositAccount;

import com.itmo.builders.CustomerBuilder;

import com.itmo.commands.AddCommand;

import com.itmo.interfaces.AbstractBank;

import com.itmo.interfaces.AbstractCentralBank;

import com.itmo.models.Bank;

import com.itmo.models.CentralBank;

import com.itmo.models.Customer;

import com.itmo.models.TimeManager;

import com.itmo.tools.BanksException;

import org.testng.Assert;

import org.testng.annotations.Test;

public class BanksTests {

private static final AbstractCentralBank MyCentralBank = new CentralBank();

private static final AbstractBank MyBank = new Bank(100000,

10000,

5,

50000,

500);

private final TimeManager timeManager = TimeManager.getTimeManager(MyCentralBank);

@Test

public void WithdrawFromDepositBeforeExpiration\_RewindTime\_WithdrawAfterExpiration\_ThrowException() throws BanksException {

MyCentralBank.addBank(MyBank);

Customer customer = new CustomerBuilder().

setName("Sergey", "Yurpalov").

setPassport("123").

setAddress("123").

build();

DepositAccount depositAccount = MyBank.createDepositAccount(customer, 10, 75000);

Assert.assertThrows(BanksException.class, () -> {

MyBank.withdrawFundsFromAccount(depositAccount, 10000);

});

timeManager.dissipateDays(11);

MyBank.withdrawFundsFromAccount(depositAccount, 10000);

Assert.assertEquals(depositAccount.getFunds(), 65000);

}

@Test

public void CreateAccounts\_UseTransactionCommands\_UndoTransactionCommands\_RewindTimeForDividends() throws BanksException {

MyCentralBank.addBank(MyBank);

Customer customer1 = new CustomerBuilder().

setName("Sergey", "Yurpalov").

build();

Customer customer2 = new CustomerBuilder().

setName("Vlad", "Povish").

setAddress("1234").

build();

Customer customer3 = new CustomerBuilder().

setName("123", "123").

setAddress("1234").

setPassport("1234")

.build();

DebitAccount debitAccount = MyBank.createDebitAccount(customer1);

CreditAccount creditAccount = MyBank.createCreditAccount(customer2);

DepositAccount depositAccount = MyBank.createDepositAccount(customer3, 10, 75000);

AddCommand addCommand1 = MyBank.addFundsToAccount(debitAccount, 5000);

Assert.assertEquals(debitAccount.getFunds(), 5000);

MyBank.undoCommand(addCommand1);

Assert.assertEquals(debitAccount.getFunds(), 0);

MyBank.addFundsToAccount(debitAccount, 5000);

MyBank.transferFundsBetweenAccounts(debitAccount, creditAccount, 2500);

Assert.assertEquals(creditAccount.getFunds(), 2500);

MyBank.withdrawFundsFromAccount(creditAccount, 9000);

timeManager.dissipateMonths(1);

Assert.assertEquals(creditAccount.getFunds(), -21500);

Assert.assertEquals(Math.round(depositAccount.getFunds()), 75216);

Assert.assertEquals(Math.round(debitAccount.getFunds()), 2510);

}

@Test

public void SurpassTransactionLimit\_RegisterUser\_SatisfyTransactionLimit\_ThrowException() throws BanksException {

MyCentralBank.addBank(MyBank);

Customer customer1 = new CustomerBuilder().

setName("Sergey", "Yurpalov").

build();

DebitAccount debitAccount = MyBank.createDebitAccount(customer1);

MyBank.addFundsToAccount(debitAccount, 20000);

Assert.assertThrows(BanksException.class, () -> {

MyBank.withdrawFundsFromAccount(debitAccount, 10000);

});

MyBank.verifyCustomer(customer1, "123", "123");

MyBank.withdrawFundsFromAccount(debitAccount, 10000);

Assert.assertEquals(debitAccount.getFunds(), 10000);

}

@Test

public void SurpassCreditLimit\_ChangeCreditLimitForBank\_SatisfyCreditLimit\_ThrowException() throws BanksException {

MyCentralBank.addBank(MyBank);

Customer customer1 = new CustomerBuilder().

setName("Sergey", "Yurpalov").

setPassport("123").

setAddress("123").

build();

CreditAccount creditAccount = MyBank.createCreditAccount(customer1);

Assert.assertThrows(BanksException.class, () -> {

MyBank.withdrawFundsFromAccount(creditAccount, 55000);

});

MyCentralBank.setNewCreditLimitAmount(60000, MyBank);

MyBank.withdrawFundsFromAccount(creditAccount, 55000);

Assert.assertEquals(creditAccount.getFunds(), -55000);

}

}

Добавлен файл src/com/itmo/builders/CustomerBuilder.java:

package com.itmo.builders;

import com.itmo.models.Customer;

import com.itmo.tools.BanksException;

public class CustomerBuilder {

private Customer customer;

public CustomerBuilder() {

customer = new Customer();

}

public CustomerBuilder setName(String firstName, String lastName) {

customer.setFirstName(firstName);

customer.setLastName(lastName);

return this;

}

public CustomerBuilder setAddress(String address) {

customer.setAddress(address);

return this;

}

public CustomerBuilder setPassport(String passport) {

customer.setPassport(passport);

return this;

}

public Customer build() throws BanksException {

if (customer.hasName()) {

return customer;

}

Refresh();

throw new BanksException("Unable to create customer without name");

}

private void Refresh() {

customer = null;

}

}

Добавлен файл src/com/itmo/commands/AbstractCommand.java:

package com.itmo.commands;

import com.itmo.tools.BanksException;

public interface AbstractCommand {

void execute() throws BanksException;

void undo() throws BanksException;

}

Добавлен файл src/com/itmo/commands/AddCommand.java:

package com.itmo.commands;

import com.itmo.accounts.AbstractAccount;

import com.itmo.tools.BanksException;

public class AddCommand implements AbstractCommand{

private final AbstractAccount account;

private final double fundsAmount;

public AddCommand(AbstractAccount account, double fundsAmount) {

this.account = account;

this.fundsAmount = fundsAmount;

}

public void execute() {

account.addFunds(fundsAmount);

}

public void undo() throws BanksException {

account.withdrawFunds(fundsAmount);

}

}

Добавлен файл src/com/itmo/commands/CommandInvoker.java:

package com.itmo.commands;

import com.itmo.tools.BanksException;

public class CommandInvoker {

private AbstractCommand currentCommand;

public void setCommand(AbstractCommand command)

{

currentCommand = command;

}

public void runCommand() throws BanksException {

if (currentCommand == null) {

throw new BanksException("No command is chosen to run");

}

currentCommand.execute();

refresh();

}

public void undoCommand() throws BanksException {

if (currentCommand == null) {

throw new BanksException("No command is chosen be undone");

}

currentCommand.undo();

refresh();

}

private void refresh() {

currentCommand = null;

}

}

Добавлен файл src/com/itmo/commands/TransferCommand.java:

package com.itmo.commands;

import com.itmo.accounts.AbstractAccount;

import com.itmo.tools.BanksException;

public class TransferCommand implements AbstractCommand {

private final AbstractAccount accountToWithdraw;

private final AbstractAccount accountToAdd;

private final double fundsAmount;

public TransferCommand(AbstractAccount accountToWithdraw, AbstractAccount accountToAdd, double fundsAmount) {

this.accountToWithdraw = accountToWithdraw;

this.accountToAdd = accountToAdd;

this.fundsAmount = fundsAmount;

}

public void execute() throws BanksException {

accountToWithdraw.withdrawFunds(fundsAmount);

accountToAdd.addFunds(fundsAmount);

}

public void undo() throws BanksException {

accountToWithdraw.addFunds(fundsAmount);

accountToAdd.withdrawFunds(fundsAmount);

}

}

Добавлен файл src/com/itmo/commands/WithdrawCommand.java:

package com.itmo.commands;

import com.itmo.accounts.AbstractAccount;

import com.itmo.tools.BanksException;

public class WithdrawCommand implements AbstractCommand {

private final AbstractAccount account;

private final double fundsAmount;

public WithdrawCommand(AbstractAccount account, double fundsAmount) {

this.account = account;

this.fundsAmount = fundsAmount;

}

public void execute() throws BanksException {

account.withdrawFunds(fundsAmount);

}

public void undo() {

account.addFunds(fundsAmount);

}

}

Добавлен файл src/com/itmo/console\_interface/Commands.java:

package com.itmo.console\_interface;

public enum Commands {

PRINT\_INFO\_CURRENT\_ACCOUNT,

ADD\_DEBIT\_ACCOUNT,

ADD\_CREDIT\_ACCOUNT,

ADD\_DEPOSIT\_ACCOUNT,

CHOOSE\_ACCOUNT,

DELETE\_ACCOUNT\_BY\_ID,

DISSIPATE\_DAYS,

DISSIPATE\_MONTHS,

ADD\_FUNDS,

WITHDRAW\_FUNDS,

TRANSFER\_FUNDS,

UNDO\_LAST\_COMMAND,

READ\_NOTIFICATIONS,

CLEAR\_NOTIFICATIONS,

DISCARD\_NOTIFICATIONS,

SET\_NEW\_TRANSACTION\_LIMIT\_FOR\_UNREGISTERED\_CUSTOMER,

SET\_NEW\_TRANSACTION\_LIMIT\_FOR\_REGISTERED\_CUSTOMER,

SET\_NEW\_DEBIT\_ANNUAL\_INTEREST\_ON\_BALANCE,

SET\_NEW\_CREDIT\_LIMIT\_AMOUNT,

SET\_NEW\_CREDIT\_FEE,

LIST\_ALL\_CUSTOMERS\_ACCOUNTS

}

Добавлен файл src/com/itmo/console\_interface/ConsoleInterface.java:

package com.itmo.console\_interface;

import com.itmo.accounts.AbstractAccount;

import com.itmo.commands.AbstractCommand;

import com.itmo.interfaces.AbstractBank;

import com.itmo.interfaces.AbstractCentralBank;

import com.itmo.interfaces.AbstractCustomer;

import com.itmo.models.TimeManager;

import com.itmo.tools.BanksException;

import java.util.\*;

import java.util.function.Consumer;

public class ConsoleInterface {

private final Map<Commands, Consumer<String[]>> commands = new HashMap<>();

private final AbstractBank bank;

private final AbstractCustomer customer;

private final AbstractCentralBank centralBank;

private final TimeManager timeManager;

private AbstractAccount account;

private AbstractCommand lastExecutedCommand;

private static final String RESET = "\u001B[0m";

private static final String RED = "\u001B[31m";

private static final String GREEN = "\u001B[32m";

private static final String PURPLE = "\u001B[35m";

public ConsoleInterface(AbstractCentralBank centralBank, AbstractBank bank, AbstractCustomer customer) {

commands.put(Commands.PRINT\_INFO\_CURRENT\_ACCOUNT, this::printInfoCurrentAccount);

commands.put(Commands.ADD\_DEBIT\_ACCOUNT, this::addDebitAccount);

commands.put(Commands.ADD\_CREDIT\_ACCOUNT, this::addCreditAccount);

commands.put(Commands.ADD\_DEPOSIT\_ACCOUNT, this::addDepositAccount);

commands.put(Commands.CHOOSE\_ACCOUNT, this::chooseAccount);

commands.put(Commands.DELETE\_ACCOUNT\_BY\_ID, this::deleteAccountById);

commands.put(Commands.DISSIPATE\_DAYS, this::dissipateDays);

commands.put(Commands.DISSIPATE\_MONTHS, this::dissipateMonths);

commands.put(Commands.ADD\_FUNDS, this::addFunds);

commands.put(Commands.WITHDRAW\_FUNDS, this::withdrawFunds);

commands.put(Commands.TRANSFER\_FUNDS, this::transferFunds);

commands.put(Commands.UNDO\_LAST\_COMMAND, this::undoLastCommand);

commands.put(Commands.READ\_NOTIFICATIONS, this::readNotifications);

commands.put(Commands.CLEAR\_NOTIFICATIONS, this::clearNotifications);

commands.put(Commands.DISCARD\_NOTIFICATIONS, this::discardNotifications);

commands.put(Commands.SET\_NEW\_TRANSACTION\_LIMIT\_FOR\_REGISTERED\_CUSTOMER, this::setNewTransactionLimitForRegisteredCustomer);

commands.put(Commands.SET\_NEW\_TRANSACTION\_LIMIT\_FOR\_UNREGISTERED\_CUSTOMER, this::setNewTransactionLimitForUnregisteredCustomer);

commands.put(Commands.SET\_NEW\_DEBIT\_ANNUAL\_INTEREST\_ON\_BALANCE, this::setNewDebitAnnualInterestOnBalance);

commands.put(Commands.SET\_NEW\_CREDIT\_LIMIT\_AMOUNT, this::setNewCreditLimitAmount);

commands.put(Commands.SET\_NEW\_CREDIT\_FEE, this::setNewCreditFee);

commands.put(Commands.LIST\_ALL\_CUSTOMERS\_ACCOUNTS, this::listAllCustomersAccounts);

this.centralBank = centralBank;

this.bank = bank;

this.customer = customer;

this.timeManager = TimeManager.getTimeManager(centralBank);

}

public void start() {

String line;

Scanner scanner = new Scanner(System.in);

while (!(line = scanner.nextLine().toUpperCase(Locale.ROOT)).equals("")) {

if (line.charAt(0) != '/' || line.length() == 1) {

System.out.println("Incorrect input - not a command");

continue;

}

if (line.split(" ")[0].equals("/EXIT")) {

printColoredMessage("Shutdown...", GREEN);

break;

}

Commands command;

try {

command = Commands.valueOf(line.substring(1).split(" ")[0]);

String[] splitLine = line.split(" ");

String[] args = Arrays.copyOfRange(splitLine, 1, splitLine.length);

executeCommand(command, args);

} catch (IllegalArgumentException e) {

printColoredMessage("Incorrect input - not a command", RED);

} catch (BanksException e) {

printColoredMessage(e.getMessage(), RED);

} catch (NullPointerException e) {

printColoredMessage("The object to act is not chosen", RED);

} catch (ArrayIndexOutOfBoundsException e){

printColoredMessage("Insufficient amount of arguments", RED);

}

}

}

private void executeCommand(Commands command, String[] args) throws BanksException {

commands.get(command).accept(args);

}

private void printColoredMessage(String message, String color) {

System.out.println(color + message + RESET);

}

private void printInfoCurrentAccount(String[] args) {

System.out.println(account.getId() + " " + account.getType() + " has " + account.getFunds());

}

private void chooseAccount(String[] args) {

account = bank.getAccountById(args[0]);

printColoredMessage("Chosen account to act with: " + account.getId(), GREEN);

}

private void addDebitAccount(String[] args) {

var account = bank.createDebitAccount(customer);

printColoredMessage("Debit account successfully created with id: " + account.getId(), GREEN);

}

private void addCreditAccount(String[] args) {

var account = bank.createCreditAccount(customer);

printColoredMessage("Credit account successfully created with id: " + account.getId(), GREEN);

}

private void addDepositAccount(String[] args) {

var account = bank.createDepositAccount(customer, Integer.parseInt(args[0]), Double.parseDouble(args[1]));

printColoredMessage("Deposit account successfully created with id: " + account.getId(), GREEN);

}

private void deleteAccountById(String[] args) {

try {

bank.removeAccount(bank.getAccountById(args[0]));

} catch (BanksException e) {

printColoredMessage(e.getMessage(), RED);

return;

}

printColoredMessage("Account with id: " + args[0] + " was successfully deleted", GREEN);

}

private void transferFunds(String[] args) {

try {

lastExecutedCommand = bank.transferFundsBetweenAccounts(

account,

bank.getAccountById(args[0]),

Double.parseDouble(args[1]));

} catch (BanksException e) {

printColoredMessage(e.getMessage(), RED);

return;

}

printColoredMessage("Funds were transferred from account with id" + account.getId() +

" to account with id: " + args[0], GREEN);

}

private void withdrawFunds(String[] args) {

try {

lastExecutedCommand = bank.withdrawFundsFromAccount(

account,

Double.parseDouble(args[0]));

} catch (BanksException e) {

printColoredMessage(e.getMessage(), RED);

return;

}

printColoredMessage("Funds were withdrawn from account with id: " + account.getId(), GREEN);

}

private void addFunds(String[] args) {

try {

lastExecutedCommand = bank.addFundsToAccount(

account,

Double.parseDouble(args[0]));

} catch (BanksException e) {

printColoredMessage(e.getMessage(), RED);

return;

}

printColoredMessage("Funds were added to account with id: " + account.getId(), GREEN);

}

private void undoLastCommand(String[] args) {

try {

bank.undoCommand(lastExecutedCommand);

} catch (BanksException e) {

printColoredMessage(e.getMessage(), RED);

return;

}

lastExecutedCommand = null;

printColoredMessage("Last command was successfully undone", GREEN);

}

private void dissipateDays(String[] args) {

timeManager.dissipateDays(Integer.parseInt(args[0]));

printColoredMessage("Successfully rewound " + args[0] + " days", PURPLE);

}

private void dissipateMonths(String[] args) {

timeManager.dissipateMonths(Integer.parseInt(args[0]));

printColoredMessage("Successfully rewound " + args[0] + " months", PURPLE);

}

private void readNotifications(String[] args) {

customer.getNotifications().forEach(System.out::println);

}

private void clearNotifications(String[] args) {

customer.clearNotifications();

}

private void discardNotifications(String[] args) {

customer.clearNotifications();

printColoredMessage("Cleared all notifications", GREEN);

}

private void setNewCreditFee(String[] args) {

centralBank.setNewCreditFee(Double.parseDouble(args[0]), bank);

printColoredMessage("New credit fee " + args[0] + " was set", GREEN);

}

private void setNewCreditLimitAmount(String[] args) {

centralBank.setNewCreditLimitAmount(Double.parseDouble(args[0]), bank);

printColoredMessage("New credit limit " + args[0] + " was set", GREEN);

}

private void setNewDebitAnnualInterestOnBalance(String[] args) {

centralBank.setNewDebitAnnualInterestOnBalance(Double.parseDouble(args[0]), bank);

printColoredMessage("New debit annual interest " + args[0] + " was set", GREEN);

}

private void setNewTransactionLimitForUnregisteredCustomer(String[] args) {

centralBank.setNewTransactionLimitForUnregisteredCustomer(Double.parseDouble(args[0]), bank);

printColoredMessage("New transaction limit for unregistered customers " + args[0] + " was set", GREEN);

}

private void setNewTransactionLimitForRegisteredCustomer(String[] args) {

centralBank.setNewTransactionLimitForRegisteredCustomer(Double.parseDouble(args[0]), bank);

printColoredMessage("New transaction limit for registered customers " + args[0] + " was set", GREEN);

}

private void listAllCustomersAccounts(String[] args) {

customer.getAccountIds().forEach(id ->

{

var account = bank.getAccountById(id.toString());

System.out.println(account.getId() + " " + account.getType() + " has " + account.getFunds());

});

}

}

Добавлен файл src/com/itmo/interfaces/AbstractBank.java:

package com.itmo.interfaces;

import com.itmo.accounts.\*;

import com.itmo.commands.AbstractCommand;

import com.itmo.commands.AddCommand;

import com.itmo.commands.TransferCommand;

import com.itmo.commands.WithdrawCommand;

import com.itmo.tools.BanksException;

import java.util.List;

public interface AbstractBank {

double getTransactionLimitForRegisteredCustomer();

double getTransactionLimitForUnregisteredCustomer();

double getDebitAnnualInterestOnBalance();

double getCreditLimitAmount();

double getCreditFee();

DebitAccount createDebitAccount(AbstractCustomer customer);

CreditAccount createCreditAccount(AbstractCustomer customer);

DepositAccount createDepositAccount(

AbstractCustomer customer,

int daysToExpire,

double funds);

void removeAccount(AbstractAccount account) throws BanksException;

void notifyAllCustomers(String message);

void notifySpecificCustomers(AccountTypes accountType, String message);

AddCommand addFundsToAccount(AbstractAccount account, double funds) throws BanksException;

WithdrawCommand withdrawFundsFromAccount(AbstractAccount account, double funds) throws BanksException;

TransferCommand transferFundsBetweenAccounts(AbstractAccount accountToWithdraw, AbstractAccount accountToAdd, double funds) throws BanksException;

void undoCommand(AbstractCommand command) throws BanksException;

void updateDay();

void updateMonth();

AbstractAccount getAccountById(String id);

List<AbstractAccount> getAllCustomerAccounts(AbstractCustomer customer);

void verifyCustomer(AbstractCustomer customer, String newPassport, String newAddress);

void changeTransactionLimitForRegisteredCustomer(double limit);

void changeTransactionLimitForUnregisteredCustomer(double limit);

void changeDebitAnnualInterestOnBalance(double interest);

void changeCreditLimitAmount(double limit);

void changeCreditFee(double fee);

}

Добавлен файл src/com/itmo/interfaces/AbstractCentralBank.java:

package com.itmo.interfaces;

import com.itmo.tools.BanksException;

public interface AbstractCentralBank {

void addBank(AbstractBank bank);

void removeBank(AbstractBank bank) throws BanksException;

void setNewTransactionLimitForUnregisteredCustomer(double limit, AbstractBank bank);

void setNewTransactionLimitForRegisteredCustomer(double limit, AbstractBank bank);

void setNewDebitAnnualInterestOnBalance(double interest, AbstractBank bank);

void setNewCreditLimitAmount(double limit, AbstractBank bank);

void setNewCreditFee(double fee, AbstractBank bank);

void notifyBanksDayPassed();

void notifyBanksMonthPassed();

}

Добавлен файл src/com/itmo/interfaces/AbstractCustomer.java:

package com.itmo.interfaces;

import java.util.List;

import java.util.UUID;

public interface AbstractCustomer {

String getFirstName();

void setFirstName(String firstName);

String getLastName();

void setLastName(String lastName);

String getAddress();

void setAddress(String address);

String getPassport();

void setPassport(String passport);

List<String> getNotifications();

List<UUID> getAccountIds();

boolean hasName();

boolean isVerified();

void update(String message);

void clearNotifications();

}

Добавлен файл src/com/itmo/models/Bank.java:

package com.itmo.models;

import com.itmo.accounts.\*;

import com.itmo.commands.\*;

import com.itmo.interfaces.AbstractBank;

import com.itmo.interfaces.AbstractCustomer;

import com.itmo.tools.BanksException;

import java.util.ArrayList;

import java.util.List;

import java.util.Objects;

import java.util.stream.Collectors;

public class Bank implements AbstractBank {

private final List<AbstractAccount> accounts = new ArrayList<>();

private final CommandInvoker invoker = new CommandInvoker();

private double transactionLimitForRegisteredCustomer;

private double transactionLimitForUnregisteredCustomer;

private double debitAnnualInterestOnBalance;

private double creditLimitAmount;

private double creditFee;

public Bank(double transactionLimitForRegisteredCustomer,

double transactionLimitForUnregisteredCustomer,

double debitAnnualInterestOnBalance,

double creditLimitAmount,

double creditFee) {

this.transactionLimitForRegisteredCustomer = transactionLimitForRegisteredCustomer;

this.transactionLimitForUnregisteredCustomer = transactionLimitForUnregisteredCustomer;

this.debitAnnualInterestOnBalance = debitAnnualInterestOnBalance;

this.creditLimitAmount = creditLimitAmount;

this.creditFee = creditFee;

}

public double getTransactionLimitForRegisteredCustomer() {

return transactionLimitForRegisteredCustomer;

}

public void setTransactionLimitForRegisteredCustomer(double limit) {

transactionLimitForRegisteredCustomer = limit;

}

public double getTransactionLimitForUnregisteredCustomer() {

return transactionLimitForUnregisteredCustomer;

}

public void setTransactionLimitForUnregisteredCustomer(double limit) {

transactionLimitForUnregisteredCustomer = limit;

}

public double getDebitAnnualInterestOnBalance() {

return debitAnnualInterestOnBalance;

}

public void setDebitAnnualInterestOnBalance(double interest) {

debitAnnualInterestOnBalance = interest;

}

public double getCreditLimitAmount() {

return creditLimitAmount;

}

public void setCreditLimitAmount(double limit) {

creditLimitAmount = limit;

}

public double getCreditFee() {

return creditFee;

}

public void setCreditFee(double fee) {

creditFee = fee;

}

public DebitAccount createDebitAccount(AbstractCustomer customer) {

var newAccount = new DebitAccount(

transactionLimitForRegisteredCustomer,

transactionLimitForUnregisteredCustomer,

debitAnnualInterestOnBalance,

customer);

accounts.add(newAccount);

return newAccount;

}

public CreditAccount createCreditAccount(AbstractCustomer customer) {

var newAccount = new CreditAccount(

customer,

transactionLimitForRegisteredCustomer,

transactionLimitForUnregisteredCustomer,

creditFee,

creditLimitAmount);

accounts.add(newAccount);

return newAccount;

}

public DepositAccount createDepositAccount(

AbstractCustomer customer,

int daysToExpire,

double startFunds) {

var newAccount = new DepositAccount(

transactionLimitForRegisteredCustomer,

transactionLimitForUnregisteredCustomer,

startFunds,

daysToExpire,

customer);

accounts.add(newAccount);

return newAccount;

}

public void removeAccount(AbstractAccount account) throws BanksException {

if (!accounts.remove(account)) {

throw new BanksException("Account to delete was not found");

}

}

public void notifyAllCustomers(String message) {

accounts.forEach(a -> a.notifyOwner(message));

}

public void notifySpecificCustomers(AccountTypes accountType, String message) {

accounts.forEach(account -> {

if (account.getType() == accountType) {

account.notifyOwner(message);

}

});

}

public void updateDay() {

accounts.forEach(AbstractAccount::updateDay);

}

public void updateMonth() {

accounts.forEach(AbstractAccount::updateMonth);

}

public AddCommand addFundsToAccount(AbstractAccount account, double funds) throws BanksException {

AddCommand addCommand = new AddCommand(account, funds);

invoker.setCommand(addCommand);

invoker.runCommand();

return addCommand;

}

public WithdrawCommand withdrawFundsFromAccount(AbstractAccount account, double funds) throws BanksException {

WithdrawCommand withdrawCommand = new WithdrawCommand(account, funds);

invoker.setCommand(withdrawCommand);

invoker.runCommand();

return withdrawCommand;

}

public TransferCommand transferFundsBetweenAccounts(

AbstractAccount accountToWithdraw,

AbstractAccount accountToAdd,

double funds) throws BanksException {

TransferCommand transferCommand = new TransferCommand(accountToWithdraw, accountToAdd, funds);

invoker.setCommand(transferCommand);

invoker.runCommand();

return transferCommand;

}

public void undoCommand(AbstractCommand command) throws BanksException {

invoker.setCommand(command);

invoker.undoCommand();

}

public AbstractAccount getAccountById(String id) {

return accounts.stream().filter(account ->

Objects.equals(account.getId().toString(), id)).findFirst().orElse(null);

}

public List<AbstractAccount> getAllCustomerAccounts(AbstractCustomer customer) {

return accounts.stream().filter(account -> account.getOwner() == customer).collect(Collectors.toList());

}

public void verifyCustomer(AbstractCustomer customer, String newPassport, String newAddress) {

customer.setAddress(newAddress);

customer.setPassport(newPassport);

customer.getAccountIds().forEach(id -> getAccountById(id.toString()).updateTransactionLimit());

}

public void changeTransactionLimitForRegisteredCustomer(double limit) {

transactionLimitForRegisteredCustomer = limit;

accounts.forEach(account -> account.setTransactionLimitForRegisteredCustomer(limit));

}

public void changeTransactionLimitForUnregisteredCustomer(double limit) {

transactionLimitForUnregisteredCustomer = limit;

accounts.forEach(account -> account.setTransactionLimitForUnregisteredCustomer(limit));

}

public void changeDebitAnnualInterestOnBalance(double interest) {

debitAnnualInterestOnBalance = interest;

accounts.forEach(account -> {

if (account instanceof DebitAccount debitAccount) {

try {

debitAccount.changeAnnualInterest(interest);

} catch (BanksException ignored) {}

}

});

}

public void changeCreditLimitAmount(double limit) {

creditLimitAmount = limit;

accounts.forEach(account -> {

if (account instanceof CreditAccount creditAccount) {

try {

creditAccount.changeCreditLimit(limit);

} catch (BanksException ignored) {}

}

});

}

public void changeCreditFee(double fee) {

creditFee = fee;

accounts.forEach(account -> {

if (account instanceof CreditAccount creditAccount) {

try {

creditAccount.changeCreditFee(fee);

} catch (BanksException ignored) {}

}

});

}

}

Добавлен файл src/com/itmo/models/CentralBank.java:

package com.itmo.models;

import com.itmo.accounts.AccountTypes;

import com.itmo.interfaces.AbstractBank;

import com.itmo.interfaces.AbstractCentralBank;

import com.itmo.tools.BanksException;

import java.util.ArrayList;

import java.util.List;

public class CentralBank implements AbstractCentralBank {

private final List<AbstractBank> banks = new ArrayList<>();

public void addBank(AbstractBank bank) {

banks.add(bank);

}

public void removeBank(AbstractBank bank) throws BanksException {

if (!banks.remove(bank)) {

throw new BanksException("Bank to delete was not found");

}

}

public void setNewTransactionLimitForUnregisteredCustomer(double limit, AbstractBank bank) {

bank.changeTransactionLimitForUnregisteredCustomer(limit);

bank.notifyAllCustomers("Transaction limit For unregistered customer was changed to " + limit);

}

public void setNewTransactionLimitForRegisteredCustomer(double limit, AbstractBank bank) {

bank.changeTransactionLimitForRegisteredCustomer(limit);

bank.notifyAllCustomers("Transaction limit For registered customer was changed to " + limit);

}

public void setNewDebitAnnualInterestOnBalance(double interest, AbstractBank bank) {

bank.changeDebitAnnualInterestOnBalance(interest);

bank.notifySpecificCustomers(

AccountTypes.DEBIT\_ACCOUNT,

"Annual interest on balance was changed to " + interest);

}

public void setNewCreditLimitAmount(double limit, AbstractBank bank) {

bank.changeCreditLimitAmount(limit);

bank.notifySpecificCustomers(

AccountTypes.CREDIT\_ACCOUNT,

"Credit limit amount was changed to " + limit);

}

public void setNewCreditFee(double fee, AbstractBank bank) {

bank.changeCreditFee(fee);

bank.notifySpecificCustomers(

AccountTypes.CREDIT\_ACCOUNT,

"Credit fee was changed to " + fee);

}

public void notifyBanksDayPassed() {

banks.forEach(AbstractBank::updateDay);

}

public void notifyBanksMonthPassed() {

banks.forEach(AbstractBank::updateMonth);

}

}

Добавлен файл src/com/itmo/models/Customer.java:

package com.itmo.models;

import com.itmo.interfaces.AbstractCustomer;

import java.util.ArrayList;

import java.util.List;

import java.util.UUID;

public class Customer implements AbstractCustomer {

private String firstName;

private String lastName;

private String address;

private String passport;

private final List<String> notifications = new ArrayList<>();

private final List<UUID> accountIds = new ArrayList<>();

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public String getPassport() {

return passport;

}

public void setPassport(String passport) {

this.passport = passport;

}

public List<String> getNotifications() {

return notifications;

}

public List<UUID> getAccountIds() {

return accountIds;

}

public boolean hasName() {

return !(firstName == null || lastName == null ||

firstName.isBlank() || lastName.isBlank());

}

public boolean isVerified() {

return !(address == null || passport == null ||

firstName.isBlank() || lastName.isBlank());

}

public void update(String message) {

notifications.add(message);

}

public void clearNotifications(){

notifications.clear();

}

}

Добавлен файл src/com/itmo/models/TimeManager.java:

package com.itmo.models;

import com.itmo.interfaces.AbstractCentralBank;

import com.itmo.tools.Config;

public class TimeManager {

private static TimeManager timeManager;

private final AbstractCentralBank centralBank;

private int daysPassed;

private TimeManager(AbstractCentralBank centralBank) {

this.centralBank = centralBank;

}

public static TimeManager getTimeManager(AbstractCentralBank centralBank) {

if (timeManager == null) {

timeManager = new TimeManager(centralBank);

}

return timeManager;

}

public void dissipateDays(int numberOfDays) {

for (int i = 0; i < numberOfDays; i++) {

daysPassed++;

centralBank.notifyBanksDayPassed();

if (daysPassed % Config.DAYS\_IN\_MONTH == 0) {

centralBank.notifyBanksMonthPassed();

}

}

}

public void dissipateMonths(int numberOfMonths) {

dissipateDays(numberOfMonths \* Config.DAYS\_IN\_MONTH);

}

}

Добавлен файл src/com/itmo/tools/BanksException.java:

package com.itmo.tools;

public class BanksException extends Exception {

public BanksException(String message) {

super(message);

}

}

Добавлен файл src/com/itmo/tools/Config.java:

package com.itmo.tools;

public class Config {

public static final int DAYS\_IN\_MONTH = 30;

public static final int DAYS\_IN\_YEAR = 365;

public static final int PERCENT\_DIVIDER = 100;

}