**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: Кошкин Михаил Сергеевич

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 banks-java/.gitignore:

# Project related fils

db\_files/\*

# Maven build directory

target

# Compiled class file

\*.class

# Log file

\*.log

# BlueJ files

\*.ctxt

# Mobile Tools for Java (J2ME)

.mtj.tmp/

\*.idea

\*.iml

# Package Files #

\*.jar

\*.war

\*.nar

\*.ear

\*.zip

\*.tar.gz

\*.rar

# virtual machine crash logs, see http://www.java.com/en/download/help/error\_hotspot.xml

hs\_err\_pid\*

Добавлен файл banks-java/src/console/banks/ConsoleInterface.java:

package console.banks;

import console.banks.services.AbstractService;

import console.banks.services.BankService;

import console.banks.services.ClientService;

import console.banks.services.MainBankService;

import console.banks.providers.IOProvider;

import console.banks.tools.InputException;

import main.banks.entities.banks.Bank;

import main.banks.entities.banks.BankBuilder;

import main.banks.entities.banks.MainBank;

import main.banks.entities.clients.BankClient;

import main.banks.entities.clients.Client;

import main.banks.entities.clients.passport.PassportInfo;

import main.banks.enums.AccountTypes;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

import java.time.Period;

import java.time.format.DateTimeFormatter;

public class ConsoleInterface {

private final IOProvider ioProvider;

private MainBank mainBank;

private AbstractService currentService;

public ConsoleInterface() {

mainBank = new MainBank();

ioProvider = new IOProvider();

currentService = new MainBankService(mainBank, ioProvider);

}

public void startExecution() {

while (true) {

var input = ioProvider

.notBlankInput()

.split(" ");

if (input[0].charAt(0) != '/') {

ioProvider.printError("Input is not a command");

continue;

}

if (input[0].equals("/e") || input[0].equals("/exit"))

return;

try {

switch (input[0]) {

case "/h", "/help" -> printGuide();

case "/a", "/available" -> printAvailable();

case "/r", "/reset" -> mainBank = new MainBank();

case "/d", "/set\_default" -> setDefaultState();

case "/add", "/add\_days" -> mainBank.getTimeProvider().addTime(

Period.ofDays(Integer.parseInt(input[1])));

case "/time" -> ioProvider.printLine(

DateTimeFormatter.ofPattern("HH:mm:ss dd/MM/yyyy")

.format(mainBank.getTimeProvider().now()));

default -> handleServiceInput(input);

}

} catch (ArrayIndexOutOfBoundsException e) {

ioProvider.printError("Incorrect command arguments");

} catch (NumberFormatException e) {

ioProvider.printError("Incorrect number input");

} catch (IllegalArgumentException e) {

ioProvider.printError("Incorrect id input");

} catch (InputException e) {

ioProvider.printError(e.getMessage() == null

? "Command does not exist"

: e.getMessage());

} catch (BanksException e) {

ioProvider.printError(e.getMessage());

}

ioProvider.printEmptyLine();

}

}

private void handleServiceInput(String[] input) throws BanksException, InputException {

switch (input[0]) {

case "/client" ->

currentService = new ClientService(

mainBank,

(input.length == 2)

? mainBank.getClient(new PassportInfo(input[1]))

: mainBank.getClients()

.stream()

.filter(u -> u.getName().equals(input[1])

&& u.getSurname().equals(input[2]))

.findFirst()

.orElseThrow(() -> new InputException("Client was not found")),

ioProvider);

case "/bank" -> currentService = new BankService(

mainBank.getBank(Integer.parseInt(input[1])),

ioProvider);

case "/main", "/main\_bank" -> currentService = new MainBankService(

mainBank,

ioProvider);

default -> currentService.handleInput(input);

}

}

private void printGuide() {

ioProvider.printLines(

"Call /a or /available to show available commands,",

"required parameters will be shown in angle brackets and optional in curly.\n",

"When creating an entity leave input empty for default.");

}

private void printAvailable() {

ioProvider.printLines(

"/e, /exit - exit the application",

"/h, /help - show help guide",

"/a, /available - show available commands",

"/r, /reset - reset the system",

"/d, /set\_default - set default state",

"/add, /add\_days <count> - add days to current date",

"/time - show current application time",

"/client <passport> || <name> <surname> - set input for client",

"/bank <number> - set input for bank",

"/main, /main\_bank - set input for main bank");

ioProvider.printEmptyLine();

currentService.printAvailable();

}

private void setDefaultState() throws BanksException {

mainBank = new MainBank();

Bank bank = new BankBuilder()

.setMaxSumForDubiousClients(BigDecimal.valueOf(1000))

.setCreditCommission(BigDecimal.valueOf(5))

.setCreditLimit(BigDecimal.valueOf(-500))

.setDebitInterest(BigDecimal.valueOf(0.5))

.build();

mainBank.registerBank(bank);

Client client = new BankClient("Mikhail", "Koshkin");

client.setBalance(1000);

client.setPassport(new PassportInfo("0123 456789"));

mainBank.addClient(client);

bank.addAccount(client, AccountTypes.DEPOSIT, BigDecimal.valueOf(10));

bank.addAccount(client, AccountTypes.DEBIT, BigDecimal.valueOf(15.5));

bank.addAccount(client, AccountTypes.CREDIT).setNotifyClient(true);

currentService = new ClientService(mainBank, client, ioProvider);

}

}

Добавлен файл banks-java/src/console/banks/Main.java:

package console.banks;

public class Main {

public static void main(String[] args) {

new ConsoleInterface().startExecution();

}

}

Добавлен файл banks-java/src/console/banks/providers/IOProvider.java:

package console.banks.providers;

import main.banks.entities.clients.passport.PassportInfo;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

import java.util.Scanner;

public class IOProvider {

private final Scanner scanner;

private static final String RESET = "\033[0m";

private static final String RED\_BOLD = "\033[1;31m";

public IOProvider() {

scanner = new Scanner(System.in);

}

public String nextString() {

return scanner.nextLine();

}

public String nextString(String fieldName) {

printInputRequirement(fieldName);

return nextString();

}

public String notBlankInput() {

while (true) {

String input = nextString();

if (!input.isBlank())

return input;

}

}

public String notBlankInput(String fieldName) {

printInputRequirement(fieldName);

return notBlankInput();

}

public BigDecimal nextDecimal() {

while (true) {

String input = nextString();

if (input.isBlank())

return BigDecimal.ZERO;

try {

return new BigDecimal(input);

} catch (NumberFormatException e) {

printError("Input is not a decimal");

}

}

}

public BigDecimal nextDecimal(String fieldName) {

printInputRequirement(fieldName);

return nextDecimal();

}

public PassportInfo nextPassport() {

while (true) {

String input = nextString();

if (input.isBlank())

return null;

try {

return new PassportInfo(input);

} catch (BanksException e) {

printError("Input is not a passport");

}

}

}

public PassportInfo nextPassport(String fieldName) {

printInputRequirement(fieldName);

return nextPassport();

}

public void printEmptyLine() {

System.out.println();

}

public void printLine(String message) {

System.out.println(message);

}

public void printLine(BigDecimal decimal) {

printLine(decimal.toString());

}

public void printLines(String... messages) {

for (var message : messages)

printLine(message);

}

public void printError(String message) {

printLine(RED\_BOLD + message + RESET);

}

private void printInputRequirement(String fieldName) {

printLine("Input " + fieldName + ":");

}

}

Добавлен файл banks-java/src/console/banks/services/AbstractService.java:

package console.banks.services;

import console.banks.tools.InputException;

import main.banks.tools.BanksException;

public interface AbstractService {

void handleInput(String[] input) throws BanksException, InputException;

void printAvailable();

}

Добавлен файл banks-java/src/console/banks/services/BankService.java:

package console.banks.services;

import console.banks.providers.IOProvider;

import console.banks.tools.InputException;

import main.banks.entities.banks.Bank;

import java.math.BigDecimal;

public record BankService(Bank bank, IOProvider ioProvider) implements AbstractService {

public void handleInput(String[] input) throws InputException {

BigDecimal outputNumber = resolveGettersInput(input);

if (outputNumber != null) {

ioProvider.printLine(outputNumber);

return;

}

handleSetters(input);

}

public void printAvailable() {

ioProvider.printLines(

"/get\_dubious\_clients\_limit, /set\_dubious\_clients\_limit <amount>",

"/get\_credit\_commission, /set\_credit\_commission <amount>",

"/get\_credit\_limit, /set\_credit\_limit <amount>",

"/get\_debit\_interest, /set\_debit\_interest <amount>");

}

private BigDecimal resolveGettersInput(String[] input) {

return switch(input[0]) {

case "/get\_dubious\_clients\_limit" -> bank.getMaxSumForDubiousClients();

case "/get\_credit\_commission" -> bank.getCreditCommission();

case "/get\_credit\_limit" -> bank.getCreditLimit();

case "/get\_debit\_interest" -> bank.getDebitInterest();

default -> null;

};

}

private void handleSetters(String[] input) throws InputException{

BigDecimal inputNumber = new BigDecimal(input[1]);

switch (input[0]) {

case "/set\_dubious\_clients\_limit" -> bank.setMaxSumForDubiousClients(inputNumber);

case "/set\_credit\_commission" -> bank.setCreditCommission(inputNumber);

case "/set\_credit\_limit" -> bank.setCreditLimit(inputNumber);

case "/set\_debit\_interest" -> bank.setDebitInterest(inputNumber);

default -> throw new InputException();

}

}

}

Добавлен файл banks-java/src/console/banks/services/ClientService.java:

package console.banks.services;

import console.banks.providers.IOProvider;

import console.banks.tools.InputException;

import main.banks.entities.accounts.Account;

import main.banks.entities.banks.Bank;

import main.banks.entities.banks.MainBank;

import main.banks.entities.clients.Client;

import main.banks.entities.clients.passport.PassportInfo;

import main.banks.enums.AccountTypes;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

import java.math.RoundingMode;

import java.util.UUID;

public record ClientService(MainBank mainBank, Client client, IOProvider ioProvider) implements AbstractService {

public void handleInput(String[] input) throws BanksException, InputException {

switch (input[0]) {

case "/add\_account" -> addAccount(input);

case "/delete\_account" -> deleteAccount(UUID.fromString(input[1]));

case "/replenish" -> replenish(

UUID.fromString(input[1]),

new BigDecimal(input[2]));

case "/transact" -> transact(

UUID.fromString(input[1]),

UUID.fromString(input[2]),

new BigDecimal(input[3]));

case "/withdraw" -> withdraw(

UUID.fromString(input[1]),

new BigDecimal(input[2]));

case "/print\_info" -> printInfo();

case "/set\_address" -> client.setAddress(input[1]);

case "/set\_passport" -> client.setPassport(new PassportInfo(input[1]));

case "/set\_balance" -> client.setBalance(new BigDecimal(input[1]));

case "/not", "/print\_notifications" -> client

.getNotifications().forEach(ioProvider::printLine);

case "/clear", "/clear\_notifications" -> client.clearNotifications();

case "/notify" -> setNotifyAccount(input, true);

case "/do\_not\_notify" -> setNotifyAccount(input, false);

default -> throw new InputException();

}

}

public void printAvailable() {

ioProvider.printLines(

"/add\_account <account\_type> <bank\_id> {amount}",

" - create account of account\_type in bank with bank\_id with {start amount}",

"/delete\_account <id> - delete account with that id",

"/replenish <id> <amount> - replenish account for stated amount",

"/transact <id> <receiver\_id> <amount> - transaction to account with receiver\_id",

"/withdraw <id> <amount> - withdraw from account",

"/print\_info - show all info about client",

"/set\_address <address>",

"/set\_passport <passport>",

"/set\_balance <amount>",

"/not, /print\_notifications - show all notifications",

"/clear, /clear\_notifications - clear all notifications",

"/notify <id> - set notifications for account with stated id",

"/do\_not\_notify <id> - set no notifications for account");

}

private void addAccount(String[] input) throws BanksException, InputException {

Bank bank = mainBank.getBank(Integer.parseInt(input[1]));

AccountTypes accountTypes = switch (input[2].toLowerCase()) {

case "debit" -> AccountTypes.DEBIT;

case "deposit" -> AccountTypes.DEPOSIT;

case "credit" -> AccountTypes.CREDIT;

default -> throw new InputException("No such account type exists");

};

var startAmount = input.length == 3

? BigDecimal.ZERO

: new BigDecimal(input[3]);

bank.addAccount(client, accountTypes, startAmount);

}

private void deleteAccount(UUID accountId) throws InputException {

getBank(accountId).deleteAccount(accountId);

}

private void replenish(UUID accountId, BigDecimal amount) throws BanksException, InputException {

Bank bank = getBank(accountId);

bank.replenish(bank.getAccount(accountId), amount);

}

private void transact(UUID senderId, UUID receiverId, BigDecimal amount) throws BanksException, InputException {

Bank bank = getBank(senderId);

Account senderAccount = bank.getAccount(senderId);

checkOwner(senderAccount);

bank.transact(senderAccount, receiverId, amount);

}

private void withdraw(UUID accountId, BigDecimal amount) throws BanksException, InputException {

Bank bank = getBank(accountId);

Account account = bank.getAccount(accountId);

checkOwner(account);

bank.withdraw(account, amount);

}

private void printInfo() {

ioProvider.printLines(

"Name: " + client().getName(),

"Surname: " + client().getSurname());

String address = client.getAddress();

ioProvider.printLine(

"Address" +

(address == null || address.isBlank()

? " not set"

: ": " + address));

PassportInfo passport = client.getPassport();

ioProvider.printLine(

"Passport" +

(passport == null

? " not set"

: ": " + passport));

ioProvider.printLine("Balance: " + client.getBalance().setScale(2, RoundingMode.CEILING));

mainBank.getBanks().forEach(b -> b

.getAccounts()

.stream()

.filter(a -> a.getClient().equals(client))

.forEach(a -> ioProvider.printLines(

"\n" + a.getType(),

"Id: " + a.getId(),

"Balance: " + a.getBalance().setScale(2, RoundingMode.CEILING),

"Notifications " + (a.getNotifyClient() ? "on" : "off"))));

}

private void setNotifyAccount(String[] input, boolean notify) throws InputException {

var accountId = UUID.fromString(input[1]);

getBank(accountId).getAccount(accountId).setNotifyClient(notify);

}

private void checkOwner(Account account) throws InputException {

if (!(account.getClient().getPassport() != null && account.getClient().equals(client)))

throw new InputException("Client is not owner of the current account");

}

private Bank getBank(UUID accountId) throws InputException {

for (var bank : mainBank.getBanks())

if (bank.getAccount(accountId) != null)

return bank;

throw new InputException("Incorrect account\_id");

}

}

Добавлен файл banks-java/src/console/banks/services/MainBankService.java:

package console.banks.services;

import console.banks.providers.IOProvider;

import console.banks.tools.InputException;

import main.banks.entities.banks.Bank;

import main.banks.entities.banks.BankBuilder;

import main.banks.entities.banks.MainBank;

import main.banks.entities.clients.BankClient;

import main.banks.entities.clients.Client;

import main.banks.entities.clients.passport.PassportInfo;

import main.banks.tools.BanksException;

public record MainBankService(MainBank mainBank, IOProvider ioProvider) implements AbstractService {

public void handleInput(String[] input) throws BanksException, InputException {

switch (input[0]) {

case "/add\_bank" -> mainBank.registerBank(createBank());

case "/add\_client" -> mainBank.addClient(createClient());

case "/delete\_bank" -> mainBank.deleteBank(mainBank

.getBank(Integer.parseInt(input[1])));

case "/delete\_client" -> mainBank.deleteClient(mainBank

.getClient(new PassportInfo(input[1])));

case "/accrue" -> mainBank.accruePercentage();

default -> throw new InputException();

}

}

public void printAvailable() {

ioProvider.printLines(

"/add\_bank - create bank and add it to system",

"/add\_client - create client and add them to system",

"/delete\_bank <number> - delete bank",

"/delete\_client <passport> - delete client",

"/accrue - accrue percents or commission");

}

private Bank createBank() {

return new BankBuilder()

.setMaxSumForDubiousClients(ioProvider.nextDecimal(

"max sum for dubious clients"))

.setCreditCommission(ioProvider.nextDecimal(

"credit commission"))

.setCreditLimit(ioProvider.nextDecimal(

"credit limit"))

.setDebitInterest(ioProvider.nextDecimal(

"debit interest"))

.build();

}

private Client createClient() throws BanksException {

var client = new BankClient(

ioProvider.notBlankInput("name"),

ioProvider.notBlankInput("surname"));

client.setBalance(ioProvider.nextDecimal("balance"));

client.setAddress(ioProvider.nextString("address"));

client.setPassport(ioProvider.nextPassport("passport info"));

return client;

}

}

Добавлен файл banks-java/src/console/banks/tools/InputException.java:

package console.banks.tools;

public class InputException extends Exception {

public InputException() {

super();

}

public InputException(String message) {

super(message);

}

}

Добавлен файл banks-java/src/main/banks/Main.java:

package main.banks;

public class Main {

public static void main(String[] args) {

}

}

Добавлен файл banks-java/src/main/banks/entities/accounts/Account.java:

package main.banks.entities.accounts;

import main.banks.entities.accounts.balance.BalanceState;

import main.banks.entities.banks.Bank;

import main.banks.entities.clients.Client;

import main.banks.enums.AccountTypes;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

import java.util.ArrayList;

import java.util.List;

import java.util.UUID;

public abstract class Account {

private final UUID id;

private final Client client;

protected final Bank bank;

private boolean notifyClient;

protected List<BalanceState> history;

protected BigDecimal balance;

protected Account(Bank bank, Client client, BigDecimal balance) throws BanksException {

this.bank = bank;

this.client = client;

this.balance = BigDecimal.ZERO;

id = UUID.randomUUID();

history = new ArrayList<>();

setBalance(balance);

}

public UUID getId() {

return id;

}

public Client getClient() {

return client;

}

public BigDecimal getBalance() {

return balance;

}

public void setBalance(BigDecimal balance) throws BanksException {

this.balance = balance;

history.add(new BalanceState(balance, bank.getTimeProvider().now()));

}

public void setNotifyClient(boolean notifyClient) {

this.notifyClient = notifyClient;

}

public boolean getNotifyClient() {

return notifyClient;

}

public List<BalanceState> getHistory() {

return history;

}

public boolean isInstanceOf(AccountTypes type) {

return switch (type) {

case DEBIT -> this instanceof DebitAccount;

case DEPOSIT -> this instanceof DepositAccount;

case CREDIT -> this instanceof CreditAccount;

};

}

public void notifyClient(String notificationMessage) {

if (notifyClient)

client.addNotification(notificationMessage);

}

public abstract void accrualOfPercents(BalanceState balanceState, int days);

public abstract AccountTypes getType();

}

Добавлен файл banks-java/src/main/banks/entities/accounts/CreditAccount.java:

package main.banks.entities.accounts;

import main.banks.entities.accounts.balance.BalanceState;

import main.banks.entities.banks.Bank;

import main.banks.entities.clients.Client;

import main.banks.enums.AccountTypes;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

public class CreditAccount extends Account {

public CreditAccount(Bank bank, Client client, BigDecimal balance) throws BanksException {

super(bank, client, balance);

}

@Override

public void setBalance(BigDecimal balance) throws BanksException {

if (balance.compareTo(bank.getCreditLimit()) < 0)

throw new BanksException("Account balance can't be less than " + bank.getCreditLimit());

super.setBalance(balance);

}

@Override

public void accrualOfPercents(BalanceState balanceState, int days) {

try {

if (balanceState.getBalance().compareTo(BigDecimal.ZERO) < 0)

setBalance(getBalance().subtract(BigDecimal.valueOf(days)

.multiply(bank.getCreditCommission())));

}

catch (BanksException e) {

balance = bank.getCreditLimit();

}

}

@Override

public AccountTypes getType() {

return AccountTypes.CREDIT;

}

}

Добавлен файл banks-java/src/main/banks/entities/accounts/DebitAccount.java:

package main.banks.entities.accounts;

import main.banks.entities.accounts.balance.BalanceState;

import main.banks.entities.banks.Bank;

import main.banks.entities.clients.Client;

import main.banks.enums.AccountTypes;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

public class DebitAccount extends Account {

public DebitAccount(Bank bank, Client client, BigDecimal balance) throws BanksException {

super(bank, client, balance);

}

@Override

public void setBalance(BigDecimal balance) throws BanksException {

if (balance.compareTo(BigDecimal.ZERO) < 0)

throw new BanksException("Account balance can't be less than 0");

super.setBalance(balance);

}

@Override

public void accrualOfPercents(BalanceState balanceState, int days) {

try {

setBalance(balance.add(balanceState.getBalance()

.multiply(bank.getDebitInterest())

.multiply(BigDecimal.valueOf(days / 365.0))));

} catch (BanksException ignored) { }

}

@Override

public AccountTypes getType() {

return AccountTypes.DEBIT;

}

}

Добавлен файл banks-java/src/main/banks/entities/accounts/DepositAccount.java:

package main.banks.entities.accounts;

import main.banks.entities.accounts.balance.BalanceState;

import main.banks.entities.banks.Bank;

import main.banks.entities.clients.Client;

import main.banks.enums.AccountTypes;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

public class DepositAccount extends Account {

private final BigDecimal interest;

public DepositAccount(Bank bank, Client client, BigDecimal balance) throws BanksException {

super(bank, client, balance);

interest = getInterest(balance);

}

@Override

public void setBalance(BigDecimal balance) throws BanksException {

if (balance.compareTo(this.balance) < 0)

throw new BanksException("Account is not withdrawable");

super.setBalance(balance);

}

public BigDecimal getInterest() {

return interest;

}

@Override

public void accrualOfPercents(BalanceState balanceState, int days) {

try {

setBalance(balance.add(balanceState.getBalance()

.multiply(interest)

.multiply(BigDecimal.valueOf(days / 365.0))));

} catch (BanksException ignored) { }

}

@Override

public AccountTypes getType() {

return AccountTypes.DEPOSIT;

}

private static BigDecimal getInterest(BigDecimal amount) {

double interest;

if (amount.compareTo(BigDecimal.valueOf(50000)) < 0)

interest = 3;

else if (amount.compareTo(BigDecimal.valueOf(100000)) < 0)

interest = 3.5;

else interest = 4;

return BigDecimal.valueOf(interest);

}

}

Добавлен файл banks-java/src/main/banks/entities/accounts/balance/BalanceState.java:

package main.banks.entities.accounts.balance;

import java.math.BigDecimal;

import java.time.OffsetDateTime;

public record BalanceState(BigDecimal balance, OffsetDateTime time) {

public BigDecimal getBalance() {

return balance;

}

public OffsetDateTime getTime() {

return time;

}

}

Добавлен файл banks-java/src/main/banks/entities/accounts/creators/AccountCreator.java:

package main.banks.entities.accounts.creators;

import main.banks.entities.accounts.Account;

import main.banks.entities.banks.Bank;

import main.banks.entities.clients.Client;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

public interface AccountCreator {

Account create(Bank bank, Client client, BigDecimal balance) throws BanksException;

}

Добавлен файл banks-java/src/main/banks/entities/accounts/creators/CreditAccountCreator.java:

package main.banks.entities.accounts.creators;

import main.banks.entities.accounts.Account;

import main.banks.entities.accounts.CreditAccount;

import main.banks.entities.banks.Bank;

import main.banks.entities.clients.Client;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

public record CreditAccountCreator() implements AccountCreator {

public Account create(Bank bank, Client client, BigDecimal balance) throws BanksException {

return new CreditAccount(bank, client, balance);

}

}

Добавлен файл banks-java/src/main/banks/entities/accounts/creators/DebitAccountCreator.java:

package main.banks.entities.accounts.creators;

import main.banks.entities.accounts.Account;

import main.banks.entities.accounts.DebitAccount;

import main.banks.entities.banks.Bank;

import main.banks.entities.clients.Client;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

public record DebitAccountCreator() implements AccountCreator {

public Account create(Bank bank, Client client, BigDecimal balance) throws BanksException {

return new DebitAccount(bank, client, balance);

}

}

Добавлен файл banks-java/src/main/banks/entities/accounts/creators/DepositAccountCreator.java:

package main.banks.entities.accounts.creators;

import main.banks.entities.accounts.Account;

import main.banks.entities.accounts.DepositAccount;

import main.banks.entities.banks.Bank;

import main.banks.entities.clients.Client;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

public record DepositAccountCreator() implements AccountCreator {

public Account create(Bank bank, Client client, BigDecimal balance) throws BanksException {

return new DepositAccount(bank, client, balance);

}

}

Добавлен файл banks-java/src/main/banks/entities/banks/Bank.java:

package main.banks.entities.banks;

import main.banks.entities.accounts.Account;

import main.banks.entities.accounts.balance.BalanceState;

import main.banks.entities.accounts.creators.AccountCreator;

import main.banks.entities.accounts.creators.CreditAccountCreator;

import main.banks.entities.accounts.creators.DebitAccountCreator;

import main.banks.entities.accounts.creators.DepositAccountCreator;

import main.banks.entities.clients.Client;

import main.banks.entities.transaction.AbstractTransaction;

import main.banks.entities.transaction.Replenishment;

import main.banks.entities.transaction.Transaction;

import main.banks.entities.transaction.Withdrawal;

import main.banks.enums.AccountTypes;

import main.banks.providers.TimeProvider;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

import java.time.OffsetDateTime;

import java.time.temporal.ChronoUnit;

import java.util.\*;

import java.util.stream.Collectors;

public class Bank {

private TimeProvider timeProvider;

private OffsetDateTime previousAccrualTime;

private BigDecimal maxSumForDubiousClients;

private BigDecimal creditCommission;

private BigDecimal creditLimit;

private BigDecimal debitInterest;

private final List<Account> accounts;

private final List<AbstractTransaction> transactions;

Bank() {

accounts = new ArrayList<>();

transactions = new ArrayList<>();

}

public List<AbstractTransaction> getTransactions() {

return transactions;

}

public BigDecimal getMaxSumForDubiousClients() {

return maxSumForDubiousClients;

}

public void setMaxSumForDubiousClients(BigDecimal maxSumForDubiousClients) {

this.maxSumForDubiousClients = maxSumForDubiousClients;

Notify("Changed max sum for transaction for untrustworthy clients to "

+ maxSumForDubiousClients);

}

public BigDecimal getCreditCommission() {

return creditCommission;

}

public void setCreditCommission(BigDecimal creditCommission) {

this.creditCommission = creditCommission;

Notify("Changed credit commission to " + creditCommission, AccountTypes.CREDIT);

}

public BigDecimal getCreditLimit() {

return creditLimit;

}

public void setCreditLimit(BigDecimal creditLimit) {

this.creditLimit = creditLimit;

Notify("Changed credit limit to " + creditLimit, AccountTypes.CREDIT);

}

public BigDecimal getDebitInterest() {

return debitInterest;

}

public void setDebitInterest(BigDecimal debitInterest) {

this.debitInterest = debitInterest;

Notify("Changed debit interest to " + debitInterest, AccountTypes.DEBIT);

}

public Account addAccount(Client client, AccountTypes accountTypes) throws BanksException {

return addAccount(client, accountTypes, BigDecimal.ZERO);

}

public TimeProvider getTimeProvider() {

return timeProvider;

}

void setTimeProvider(TimeProvider timeProvider) {

this.timeProvider = timeProvider;

previousAccrualTime = timeProvider.now();

}

public Account addAccount(Client client, AccountTypes accountType, BigDecimal balance) throws BanksException {

client.setBalance(client.getBalance().subtract(balance));

Account account = resolveAccountCreator(accountType).create(this, client, balance);

accounts.add(account);

return account;

}

public Account getAccount(UUID accountId) {

return accounts

.stream()

.filter(a -> a.getId().equals(accountId))

.findFirst()

.orElse(null);

}

public List<Account> getAccounts() {

return accounts;

}

public List<Account> findAccounts(Client client) {

return accounts.stream().filter(u -> u.getClient().equals(client)).toList();

}

public void deleteAccount(Account account) {

accounts.remove(account);

}

public void deleteAccount(UUID id) {

deleteAccount(accounts

.stream()

.filter(a -> a.getId().equals(id))

.findFirst()

.orElse(null));

}

public void replenish(Account account, BigDecimal amount) throws BanksException {

transactions.add(new Replenishment(account, amount).execute());

}

public void transact(Account sender, Account receiver, BigDecimal amount) throws BanksException {

checkIfDubious(sender, amount);

transactions.add(new Transaction(sender, receiver, amount).execute());

}

public void transact(Account sender, UUID receiverId, BigDecimal amount) throws BanksException {

Account receiver = accounts

.stream()

.filter(a -> a.getId().equals(receiverId))

.findFirst()

.orElseThrow(() -> new BanksException(

"Account with id " + receiverId + " was not found"));

transact(sender, receiver, amount);

}

public void withdraw(Account account, BigDecimal amount) throws BanksException {

checkIfDubious(account, amount);

transactions.add(new Withdrawal(account, amount).execute());

}

void accrualOfInterestOrCommission(OffsetDateTime currentTime) {

for (var account : accounts)

accrueInterestForAccount(account, currentTime);

previousAccrualTime = currentTime;

}

private static AccountCreator resolveAccountCreator(AccountTypes accountType) throws BanksException {

if (accountType == null)

throw new BanksException("Passed enum is null");

return switch (accountType) {

case DEBIT -> new DebitAccountCreator();

case DEPOSIT -> new DepositAccountCreator();

case CREDIT -> new CreditAccountCreator();

};

}

private void accrueInterestForAccount(Account account, OffsetDateTime currentTime) {

account.getHistory().add(new BalanceState(account.getBalance(), currentTime));

List<BalanceState> balanceStates = new ArrayList<>();

account.getHistory()

.stream()

.filter(b -> !previousAccrualTime.isAfter(b.getTime())

&& !b.getTime().isAfter(currentTime))

.collect(Collectors.groupingBy(b -> b.getTime().getDayOfMonth()))

.values()

.forEach(b -> balanceStates.add(b.get(b.size() - 1)));

for (int i = 1; i < balanceStates.size(); i++) {

var days = (int)ChronoUnit.DAYS.between(

balanceStates.get(i - 1).getTime(),

balanceStates.get(i).getTime());

account.accrualOfPercents(balanceStates.get(i), days);

}

}

private void checkIfDubious(Account account, BigDecimal amount) throws BanksException {

if (account == null)

throw new BanksException("Account is null");

if (amount.compareTo(maxSumForDubiousClients) > 0 && account.getClient().checkIfDubious())

throw new BanksException("Client is untrustworthy");

}

private void Notify(String notificationMessage) {

accounts.forEach(u -> u.notifyClient(notificationMessage));

}

private void Notify(String notificationMessage, AccountTypes accountType) {

accounts.stream()

.filter(a -> a.isInstanceOf(accountType))

.forEach(a -> a.notifyClient(notificationMessage));

}

}

Добавлен файл banks-java/src/main/banks/entities/banks/BankBuilder.java:

package main.banks.entities.banks;

import java.math.BigDecimal;

public class BankBuilder {

private final Bank bank;

public BankBuilder() {

bank = new Bank();

}

public BankBuilder setMaxSumForDubiousClients(BigDecimal maxSum) {

bank.setMaxSumForDubiousClients(maxSum);

return this;

}

public BankBuilder setCreditCommission(BigDecimal commission) {

bank.setCreditCommission(commission);

return this;

}

public BankBuilder setCreditLimit(BigDecimal limit) {

bank.setCreditLimit(limit);

return this;

}

public BankBuilder setDebitInterest(BigDecimal interest) {

bank.setDebitInterest(interest);

return this;

}

public Bank build() {

return bank;

}

}

Добавлен файл banks-java/src/main/banks/entities/banks/MainBank.java:

package main.banks.entities.banks;

import main.banks.entities.clients.Client;

import main.banks.entities.clients.passport.PassportInfo;

import main.banks.providers.TimeProvider;

import main.banks.tools.BanksException;

import java.time.OffsetDateTime;

import java.util.ArrayList;

import java.util.List;

public class MainBank {

private final TimeProvider timeProvider;

private final List<Bank> banks;

private final List<Client> clients;

public MainBank() {

timeProvider = new TimeProvider();

banks = new ArrayList<>();

clients = new ArrayList<>();

}

public Bank registerBank(Bank bank) {

bank.setTimeProvider(timeProvider);

banks.add(bank);

return bank;

}

public Bank registerBank(BankBuilder bankBuilder) {

return registerBank(bankBuilder.build());

}

public Bank getBank(int id) throws BanksException {

if (id < 0 || id >= banks.size())

throw new BanksException("Incorrect bank id");

return banks.get(id);

}

public void deleteBank(Bank bank) {

banks.remove(bank);

}

public List<Bank> getBanks() {

return banks;

}

public List<Client> getClients() {

return clients;

}

public Client addClient(Client client) {

clients.add(client);

return client;

}

public List<Client> findClients(String name, String surname) {

return clients

.stream()

.filter(c -> c.getName().equals(name)

&& c.getSurname().equals(surname))

.toList();

}

public Client getClient(PassportInfo passportInfo) throws BanksException {

if (passportInfo == null)

throw new BanksException("Passport info is null");

return clients

.stream()

.filter(c -> passportInfo.equals(c.getPassport()))

.findFirst()

.orElseThrow(() -> new BanksException("No client found"));

}

public void deleteClient(Client client) {

clients.remove(client);

banks.forEach(b -> b.findAccounts(client)

.forEach(a -> b.getAccounts().remove(a)));

}

public void accruePercentage() {

OffsetDateTime time = timeProvider.now();

for (var bank : banks)

bank.accrualOfInterestOrCommission(time);

}

public TimeProvider getTimeProvider() {

return timeProvider;

}

}

Добавлен файл banks-java/src/main/banks/entities/clients/BankClient.java:

package main.banks.entities.clients;

import main.banks.entities.clients.passport.PassportInfo;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

import java.util.ArrayList;

import java.util.List;

import java.util.Objects;

public class BankClient implements Client {

private final String name;

private final String surname;

private String address;

private PassportInfo passport;

private BigDecimal balance;

private List<String> notifications;

public BankClient(String name, String surname) {

this.name = name;

this.surname = surname;

notifications = new ArrayList<>();

}

public BankClient(String name, String surname, String address, PassportInfo passport) {

this(name, surname);

this.address = address;

this.passport = passport;

}

public String getName() {

return name;

}

public String getSurname() {

return surname;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public PassportInfo getPassport() {

return passport;

}

public void setPassport(PassportInfo passport) {

this.passport = passport;

}

public BigDecimal getBalance() {

return balance;

}

public void setBalance(BigDecimal balance) throws BanksException {

if (balance.compareTo(BigDecimal.ZERO) < 0)

throw new BanksException("User's balance can not be less than 0");

this.balance = balance;

}

public void setBalance(int balance) throws BanksException {

setBalance(BigDecimal.valueOf(balance));

}

public List<String> getNotifications() {

return notifications;

}

public void addNotification(String notification) {

notifications.remove(notification);

notifications.add(notification);

}

public void clearNotifications() {

notifications.clear();

}

public boolean checkIfDubious() {

return address == null || address.isBlank() || passport == null;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof BankClient that)) return false;

return passport.equals(that.passport);

}

@Override

public int hashCode() {

return Objects.hash(passport);

}

}

Добавлен файл banks-java/src/main/banks/entities/clients/Client.java:

package main.banks.entities.clients;

import main.banks.entities.clients.passport.PassportInfo;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

import java.util.List;

public interface Client {

String getName();

String getSurname();

String getAddress();

void setAddress(String address);

PassportInfo getPassport();

void setPassport(PassportInfo passport);

BigDecimal getBalance();

void setBalance(BigDecimal balance) throws BanksException;

void setBalance(int balance) throws BanksException;

List<String> getNotifications();

void addNotification(String notification);

void clearNotifications();

boolean checkIfDubious();

}

Добавлен файл banks-java/src/main/banks/entities/clients/passport/PassportInfo.java:

package main.banks.entities.clients.passport;

import java.util.Objects;

import java.util.regex.Pattern;

import main.banks.tools.BanksException;

public class PassportInfo {

private final int batch;

private final int code;

public PassportInfo(String passportInfo) throws BanksException {

var matcher = Pattern.compile("^(\\d{4})(\\d{6})$")

.matcher(passportInfo.replace(" ", ""));

if (!matcher.find())

throw new BanksException("Incorrect passport information");

batch = Integer.parseInt(matcher.group(1));

code = Integer.parseInt(matcher.group(2));

}

public int getBatch() {

return batch;

}

public int getCode() {

return code;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof PassportInfo that)) return false;

return batch == that.batch && code == that.code;

}

@Override

public int hashCode() {

return Objects.hash(batch, code);

}

@Override

public String toString() {

return String.format("%04d %06d", batch, code);

}

}

Добавлен файл banks-java/src/main/banks/entities/transaction/AbstractTransaction.java:

package main.banks.entities.transaction;

import main.banks.entities.accounts.Account;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

public abstract class AbstractTransaction {

private boolean cancelled;

protected final Account account;

protected final BigDecimal amount;

protected AbstractTransaction(Account account, BigDecimal amount) {

this.account = account;

this.amount = amount;

}

public Account getAccount() {

return account;

}

public BigDecimal getAmount() {

return amount;

}

public boolean isCancelled() {

return cancelled;

}

public abstract AbstractTransaction execute() throws BanksException;

public void cancel() throws BanksException {

if (this.cancelled)

throw new BanksException("Transaction is already cancelled");

this.cancelled = true;

}

}

Добавлен файл banks-java/src/main/banks/entities/transaction/Replenishment.java:

package main.banks.entities.transaction;

import main.banks.entities.accounts.Account;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

public class Replenishment extends AbstractTransaction {

public Replenishment(Account account, BigDecimal amount) {

super(account, amount);

}

@Override

public AbstractTransaction execute() throws BanksException {

account.getClient().setBalance(account.getClient().getBalance().subtract(amount));

account.setBalance(account.getBalance().add(amount));

return this;

}

@Override

public void cancel() throws BanksException {

super.cancel();

new Withdrawal(account, amount).execute();

}

}

Добавлен файл banks-java/src/main/banks/entities/transaction/Transaction.java:

package main.banks.entities.transaction;

import main.banks.entities.accounts.Account;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

public class Transaction extends AbstractTransaction {

private final Account receiver;

public Transaction(Account sender, Account receiver, BigDecimal amount) {

super(sender, amount);

this.receiver = receiver;

}

public Account getReceiver() {

return receiver;

}

@Override

public AbstractTransaction execute() throws BanksException {

if (account == receiver)

throw new BanksException("Can not transact to sender");

account.setBalance(account.getBalance().subtract(amount));

receiver.setBalance(receiver.getBalance().add(amount));

return this;

}

@Override

public void cancel() throws BanksException {

super.cancel();

new Transaction(receiver, account, amount).execute();

}

}

Добавлен файл banks-java/src/main/banks/entities/transaction/Withdrawal.java:

package main.banks.entities.transaction;

import main.banks.entities.accounts.Account;

import main.banks.tools.BanksException;

import java.math.BigDecimal;

public class Withdrawal extends AbstractTransaction {

public Withdrawal(Account account, BigDecimal amount) {

super(account, amount);

}

@Override

public AbstractTransaction execute() throws BanksException {

account.setBalance(account.getBalance().subtract(amount));

account.getClient().setBalance(account.getClient().getBalance().add(amount));

return this;

}

@Override

public void cancel() throws BanksException {

super.cancel();

new Replenishment(account, amount).execute();

}

}

Добавлен файл banks-java/src/main/banks/enums/AccountActions.java:

package main.banks.enums;

public enum AccountActions {

REPLENISH,

TRANSACT,

WITHDRAW

}

Добавлен файл banks-java/src/main/banks/enums/AccountTypes.java:

package main.banks.enums;

public enum AccountTypes {

DEBIT,

DEPOSIT,

CREDIT

}

Добавлен файл banks-java/src/main/banks/providers/TimeProvider.java:

package main.banks.providers;

import java.time.OffsetDateTime;

import java.time.Period;

import java.time.ZoneOffset;

public class TimeProvider {

private Period timeShift = Period.ZERO;

public OffsetDateTime now() {

return OffsetDateTime.now(ZoneOffset.UTC).plus(timeShift);

}

public void addTime(Period timeShift) {

this.timeShift = this.timeShift.plus(timeShift);

}

public Period getTimeShift() {

return timeShift;

}

}

Добавлен файл banks-java/src/main/banks/tools/BanksException.java:

package main.banks.tools;

public class BanksException extends Exception{

public BanksException(String message) {

super(message);

}

}

Добавлен файл banks-java/src/test/banks/BanksTest.java:

package test.banks;

import main.banks.entities.accounts.Account;

import main.banks.entities.banks.Bank;

import main.banks.entities.banks.BankBuilder;

import main.banks.entities.banks.MainBank;

import main.banks.entities.clients.BankClient;

import main.banks.entities.clients.Client;

import main.banks.entities.clients.passport.PassportInfo;

import main.banks.enums.AccountTypes;

import main.banks.tools.BanksException;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import java.math.BigDecimal;

import java.time.Period;

import static org.junit.jupiter.api.Assertions.\*;

class BanksTest {

private MainBank mainBank;

private final BigDecimal maxSumForDubiousClients = BigDecimal.valueOf(1000);

private final BigDecimal creditCommission = BigDecimal.valueOf(5);

private final BigDecimal creditLimit = BigDecimal.valueOf(-500);

private final BigDecimal debitInterest = BigDecimal.valueOf(0.5);

private final int dayShift = 15;

@BeforeEach

void setUp() {

mainBank = new MainBank();

mainBank.registerBank(new BankBuilder()

.setMaxSumForDubiousClients(maxSumForDubiousClients)

.setCreditCommission(creditCommission)

.setCreditLimit(creditLimit)

.setDebitInterest(debitInterest)

.build());

}

@Test

void transactionBetweenAccountsMoneyTransacted() throws BanksException {

Bank bank = mainBank.getBank(0);

Client client = new BankClient("Mikhail", "Koshkin");

client.setBalance(1000);

mainBank.addClient(client);

final var openingBalance1 = BigDecimal.valueOf(30);

final var openingBalance2 = BigDecimal.valueOf(40);

final var transactionAmount = BigDecimal.valueOf(15.5);

Account account1 = bank.addAccount(client, AccountTypes.CREDIT, openingBalance1);

Account account2 = bank.addAccount(client, AccountTypes.DEBIT, openingBalance2);

bank.transact(account1, account2, transactionAmount);

assertEquals(account1.getBalance(), openingBalance1.subtract(transactionAmount));

assertEquals(account2.getBalance(), openingBalance2.add(transactionAmount));

}

@Test

void transactionCancelledMoneyReturned() throws BanksException {

Bank bank = mainBank.getBank(0);

Client client = new BankClient("Some", "Dude");

client.setBalance(100);

mainBank.addClient(client);

final var openingBalance1 = BigDecimal.valueOf(30);

final var openingBalance2 = BigDecimal.valueOf(50);

final var transactionAmount = BigDecimal.valueOf(15.5);

Account account1 = bank.addAccount(client, AccountTypes.CREDIT, openingBalance1);

Account account2 = bank.addAccount(client, AccountTypes.DEBIT, openingBalance2);

bank.transact(account1, account2, transactionAmount);

bank.getTransactions().get(0).cancel();

assertTrue(account1.getBalance().compareTo(openingBalance1) == 0);

assertTrue(account2.getBalance().compareTo(openingBalance2) == 0);

}

@Test

void withdrawalReplenishmentMoneyChanged() throws BanksException {

Bank bank = mainBank.getBank(0);

Client client = new BankClient("Mikhail", "Koshkin");

client.setBalance(1000);

mainBank.addClient(client);

final var openingBalance = BigDecimal.valueOf(10);

final var replenishmentAmount = BigDecimal.valueOf(30);

final var withdrawalAmount = BigDecimal.valueOf(40);

Account account = bank.addAccount(client, AccountTypes.DEBIT, openingBalance);

bank.replenish(account, replenishmentAmount);

bank.withdraw(account, withdrawalAmount);

assertEquals(account.getBalance(), openingBalance.add(replenishmentAmount).subtract(withdrawalAmount));

}

@Test

void creditAccountCommissionWithNegativeBalance() throws BanksException {

Bank bank = mainBank.getBank(0);

Client client = new BankClient("Mikhail", "Koshkin");

client.setBalance(1000);

mainBank.addClient(client);

final var openingBalance = BigDecimal.valueOf(10);

final var withdrawalAmount = BigDecimal.valueOf(50);

Account account = bank.addAccount(client, AccountTypes.CREDIT, openingBalance);

bank.withdraw(account, withdrawalAmount);

mainBank.getTimeProvider().addTime(Period.ofDays(dayShift));

mainBank.accruePercentage();

assertEquals(account.getBalance(),

openingBalance

.subtract(withdrawalAmount)

.subtract(creditCommission.multiply(

BigDecimal.valueOf(dayShift))));

}

@Test

public void debitAccountInterestAccrual() throws BanksException {

Bank bank = mainBank.getBank(0);

Client client = new BankClient("Mikhail", "Koshkin");

client.setBalance(1000);

mainBank.addClient(client);

final var openingBalance = BigDecimal.valueOf(10);

Account account = bank.addAccount(client, AccountTypes.DEBIT, openingBalance);

mainBank.getTimeProvider().addTime(Period.ofDays(dayShift));

mainBank.accruePercentage();

assertEquals(account.getBalance(),

openingBalance.add(openingBalance

.multiply(debitInterest)

.multiply(BigDecimal.valueOf(dayShift / 365.0))));

}

@Test

void depositAccountInterestAccrual() throws BanksException {

Bank bank = mainBank.getBank(0);

Client client = new BankClient("Mikhail", "Koshkin");

client.setBalance(1000);

mainBank.addClient(client);

final var openingBalance = BigDecimal.valueOf(10);

final var depositInterest = BigDecimal.valueOf(3);

Account account = bank.addAccount(client, AccountTypes.DEPOSIT, openingBalance);

mainBank.getTimeProvider().addTime(Period.ofDays(dayShift));

mainBank.accruePercentage();

assertTrue(0 == account.getBalance().compareTo(

openingBalance.add(openingBalance

.multiply(depositInterest)

.multiply(BigDecimal.valueOf(dayShift / 365.0)))));

}

@Test

void creditLimitChangedClientNotified() throws BanksException {

Bank bank = mainBank.getBank(0);

Client client = new BankClient("Mikhail", "Koshkin");

client.setBalance(1000);

mainBank.addClient(client);

Account account = bank.addAccount(client, AccountTypes.CREDIT);

account.setNotifyClient(true);

bank.setCreditLimit(BigDecimal.valueOf(-200));

assertEquals(client.getNotifications().size(), 1);

}

@Test

public void ClientIsUntrustworthyCannotDoTheWithdrawal\_ThrowException() throws BanksException {

Bank bank = mainBank.getBank(0);

Client client = new BankClient("Mikhail", "Koshkin");

client.setBalance(2000);

mainBank.addClient(client);

final var openingBalance = BigDecimal.valueOf(1500);

final var withdrawalAmount = BigDecimal.valueOf(1200);

Account account = bank.addAccount(client, AccountTypes.DEBIT, openingBalance);

assertThrows(BanksException.class, () ->

bank.withdraw(account, withdrawalAmount));

client.setAddress("Some address");

client.setPassport(new PassportInfo("4444 555555"));

bank.withdraw(account, withdrawalAmount);

}

@Test

void cannotWithdrawFromDepositAccount\_ThrowException() throws BanksException {

Bank bank = mainBank.getBank(0);

Client client = new BankClient("Mikhail", "Koshkin");

client.setBalance(2000);

mainBank.addClient(client);

Account account = bank.addAccount(client, AccountTypes.DEPOSIT, BigDecimal.valueOf(20));

assertThrows(BanksException.class, () ->

bank.withdraw(account, BigDecimal.valueOf(10)));

}

@Test

void CannotHaveNegativeBalanceOnDebit\_ThrowException() throws BanksException {

Bank bank = mainBank.getBank(0);

Client client = new BankClient("Mikhail", "Koshkin");

client.setBalance(2000);

mainBank.addClient(client);

final var openingBalance = BigDecimal.valueOf(200);

final var withdrawalAmount = BigDecimal.valueOf(300);

Account account = bank.addAccount(client, AccountTypes.DEBIT, openingBalance);

assertThrows(BanksException.class, () ->

bank.withdraw(account, withdrawalAmount));

}

}