Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Лабороторная работа №1

Выполнил студент:

Моторина Евгения Викторовна
Группа: M32051

Проверил:

Чикишев Константин Максимович

Санкт-Петербург,
2022 г.

1.1. Текст задания

1 лабораторная
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
Кредитный счет – имеет кредитный лимит, в рамках которого можно уходить в минус (в плюс тоже можно). Процента на остаток нет. Есть фиксированная комиссия за использование, если клиент в минусе.
Комиссии
Периодически банки проводят операции по выплате процентов и вычету комиссии. Это значит, что нужен механизм проматывания времени, чтобы посмотреть, что будет через день/месяц/год и т.п.
Процент на остаток начисляется ежедневно от текущей суммы в этот день, но выплачивается раз в месяц (и для дебетовой карты и для депозита). Например, 3.65
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

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

1.2. Решение

Листинг 1.1: Main.java

```
1  package ru.itmo;
2
3  import ru.itmo.bank.Bank;
4  import ru.itmo.bank.CentralBank;
5  import ru.itmo.bank.ICentralBank;
6  import ru.itmo.client.Client;
7  import ru.itmo.service.ConsoleInterface;
8  import ru.itmo.service.IConsoleInterface;
9  import ru.itmo.tools.BanksException;
10
11 import java.util.Date;
12
13 public class Main {
14     public static void main(String[] args) throws BanksException {
15         ICentralBank centralBank = new CentralBank();
16
17         // Create main bank
18         Bank bank = new Bank("Tinkoff", 9.7, 4, 7000, 1500,
19                 200000);
20         centralBank.registerNewBank(bank);
21         Date startTime = new Date();
22         IConsoleInterface consoleInterface = new ConsoleInterface(
       centralBank, bank);
23         Client client = consoleInterface.greeting();
24         while (true) {
25             consoleInterface.workingWithAccount(client);
26         }
27     }
28 }
```

Листинг 1.2: CreditAccount.java

```java
package ru.itmo.account;

import ru.itmo.bank.Bank;
import ru.itmo.bank.Transaction;
import ru.itmo.client.Client;
import ru.itmo.tools.BanksException;

import java.util.ArrayList;
import java.util.UUID;

public class CreditAccount implements IAccount {
    private UUID id;
    private Bank bank;
    private Client client;
    private double size;
    private double commission;
    private double limit;
    private boolean status;

    public CreditAccount(Bank bank, Client client, double size) {
        id = UUID.randomUUID();
        this.bank = bank;
        this.client = client;
        this.size = size;
        commission = bank.getCommission();
        limit = bank.getLimit();
        status = false;
    }

    public Transaction withdrawMoney(double money) throws
    BanksException {
        Transaction newTransaction = new Transaction(id, -money);
        checkAccountStatus(money);
        if (!(size - money > (-1) * limit)) {
            throw new BanksException("The credit limit reached.");
        }
        size -= money;
        bank.getTransactions().add(newTransaction);
        return newTransaction;
    }

    public Transaction appendMoney(double money) throws BanksException
    {
        Transaction newTransaction = new Transaction(id, money);
        checkAccountStatus(money);
        size += money;
        bank.getTransactions().add(newTransaction);
        return newTransaction;
    }
```

```
48
49      public Transaction transferMoney(UUID accountId, double money)
    throws BanksException {
50          Transaction newTransaction = new Transaction(id, accountId, -
    money);
51          checkAccountStatus(money);
52          size -= money;
53          IAccount newAccount = bank.getAccountById(accountId);
54          newAccount.appendMoney(money);
55          bank.getTransactions().add(newTransaction);
56          return newTransaction;
57      }
58
59      public double updateAccountSize(boolean check) {
60          if (size < 0) {
61              size -= commission;
62          }
63          return size;
64      }
65
66      private void checkAccountStatus(double money) throws
    BanksException {
67          if (!status & money > bank.getMaximumAvailableAmount()) {
68              throw new BanksException("The account is doubtful, the
    operation is impossible.");
69          }
70      }
71
72      public UUID getId() {
73          return id;
74      }
75
76      public void setId(UUID id) {
77          this.id = id;
78      }
79
80      public Bank getBank() {
81          return bank;
82      }
83
84      public void setBank(Bank bank) {
85          this.bank = bank;
86      }
87
88      public Client getClient() {
89          return client;
90      }
91
92      public void setClient(Client client) {
93          this.client = client;
```

```
94      }
95
96      public double getSize() {
97          return size;
98      }
99
100     public void setSize(double size) {
101         this.size = size;
102     }
103
104     public double getCommission() {
105         return commission;
106     }
107
108     public void setCommission(double commission) {
109         this.commission = commission;
110     }
111
112     public double getLimit() {
113         return limit;
114     }
115
116     public void setLimit(double limit) {
117         this.limit = limit;
118     }
119
120     public boolean isStatus() {
121         return status;
122     }
123
124     public void setStatus(boolean status) {
125         this.status = status;
126     }
127 }
```

Листинг 1.3: DebitAccount.java

```java
package ru.itmo.account;

import ru.itmo.bank.Bank;
import ru.itmo.bank.Transaction;
import ru.itmo.client.Client;
import ru.itmo.tools.BanksException;

import java.util.ArrayList;
import java.util.UUID;

public class DebitAccount implements IAccount {
    private UUID id;
    private Bank bank;
    private Client client;
    private double size;
    private double percent;
    private double accumulatedSum;
    private boolean status;

    public DebitAccount(Bank bank, Client client, double size) {
        id = UUID.randomUUID();
        this.bank = bank;
        this.client = client;
        this.size = size;
        percent = bank.getPercent();
        accumulatedSum = 0;
        status = false;
    }

    public Transaction withdrawMoney(double money) throws
    BanksException {
        Transaction newTransaction = new Transaction(id, -money);
        checkAccountStatus(money);
        if (size < money) {
            throw new BanksException("Insufficient money in the
    account.");
        }
        size -= money;
        bank.getTransactions().add(newTransaction);
        return newTransaction;
    }

    public Transaction appendMoney(double money) {
        Transaction newTransaction = new Transaction(id, money);
        size += money;
        bank.getTransactions().add(newTransaction);
        return newTransaction;
    }

```

```
48      public Transaction transferMoney(UUID accountId, double money)
    throws BanksException {
49          Transaction newTransaction = new Transaction(id, accountId, -
    money);
50          checkAccountStatus(money);
51          if (size < money) {
52              throw new BanksException("Not enough money in the account.
    ");
53          }
54          size -= money;
55          IAccount newAccount = bank.getAccountById(accountId);
56          newAccount.appendMoney(money);
57          bank.getTransactions().add(newTransaction);
58          return newTransaction;
59      }
60
61      public double updateAccountSize(boolean check) {
62          if (check) {
63              size += accumulatedSum;
64              accumulatedSum = 0;
65          } else {
66              accumulatedSum += size * (percent / 100 / 365);
67          }
68          return size;
69      }
70
71      private void checkAccountStatus(double money) throws
    BanksException {
72          if (!status & money > bank.getMaximumAvailableAmount()) {
73              throw new BanksException("The account is doubtful, the
    operation is impossible.");
74          }
75      }
76
77      public UUID getId() {
78          return id;
79      }
80
81      public void setId(UUID id) {
82          this.id = id;
83      }
84
85      public Bank getBank() {
86          return bank;
87      }
88
89      public void setBank(Bank bank) {
90          this.bank = bank;
91      }
92
```

```
93      @Override
94      public Client getClient() {
95          return client;
96      }
97
98      public void setClient(Client client) {
99          this.client = client;
100     }
101
102     @Override
103     public double getSize() {
104         return size;
105     }
106
107     public void setSize(double size) {
108         this.size = size;
109     }
110
111     public double getPercent() {
112         return percent;
113     }
114
115     public void setPercent(double percent) {
116         this.percent = percent;
117     }
118
119     public double getAccumulatedSum() {
120         return accumulatedSum;
121     }
122
123     public void setAccumulatedSum(double accumulatedSum) {
124         this.accumulatedSum = accumulatedSum;
125     }
126
127     public boolean isStatus() {
128         return status;
129     }
130
131     public void setStatus(boolean status) {
132         this.status = status;
133     }
134 }
```

Листинг 1.4: DepositAccount.java

```
1  package ru.itmo.account;
2
3  import ru.itmo.bank.Bank;
4  import ru.itmo.bank.Transaction;
5  import ru.itmo.client.Client;
6  import ru.itmo.tools.BanksException;
7
8  import java.util.Date;
9  import java.util.UUID;
10
11 public class DepositAccount implements IAccount {
12     private UUID id;
13     private Bank bank;
14     private Client client;
15     private double size;
16     private double percent;
17     private Date term;
18     private double accumulatedSum;
19     private boolean status;
20
21     public DepositAccount(Bank bank, Client client, double size) {
22         id = UUID.randomUUID();
23         this.bank = bank;
24         this.client = client;
25         this.size = size;
26         percent = bank.getPercent();
27         accumulatedSum = 0;
28         status = false;
29     }
30
31     public UUID getId(IAccount account) {
32         return account.getId();
33     }
34
35     public Transaction withdrawMoney(double money) throws
   BanksException {
36         Transaction newTransaction = new Transaction(id, -money);
37         checkAccountStatus(money);
38         Date currentDate = new Date();
39         if (currentDate.before(term)) {
40             throw new BanksException("The account term is not over yet
   .");
41         }
42         size -= money;
43         bank.getTransactions().add(newTransaction);
44         return newTransaction;
45     }
46
47     public Transaction appendMoney(double money) {
```

```
48          Transaction newTransaction = new Transaction(id, money);
49          size += money;
50          bank.getTransactions().add(newTransaction);
51          return newTransaction;
52      }
53
54      public Transaction transferMoney(UUID accountId, double money)
    throws BanksException {
55          Transaction newTransaction = new Transaction(id, accountId, -
    money);
56          checkAccountStatus(money);
57          Date currentDate = new Date();
58          if (currentDate.before(term)) {
59              throw new BanksException("The account term is not over yet
    .");
60          }
61          if (size < money) {
62              throw new BanksException("Not enough money in the account.
    ");
63          }
64          size -= money;
65          IAccount newAccount = bank.getAccountById(accountId);
66          newAccount.appendMoney(money);
67          bank.getTransactions().add(newTransaction);
68          return newTransaction;
69      }
70
71      public double updateAccountSize(boolean check) {
72          if (check) {
73              size += accumulatedSum;
74              accumulatedSum = 0;
75          } else {
76              accumulatedSum += size * (percent / 100 / 365);
77          }
78          return size;
79      }
80
81      private void checkAccountStatus(double money) throws
    BanksException {
82          if (!status & money > bank.getMaximumAvailableAmount()) {
83              throw new BanksException("The account is doubtful, the
    operation is impossible.");
84          }
85      }
86
87      public UUID getId() {
88          return id;
89      }
90
91      public void setId(UUID id) {
```

```
92          this.id = id;
93      }
94
95      public Bank getBank() {
96          return bank;
97      }
98
99      public void setBank(Bank bank) {
100         this.bank = bank;
101     }
102
103     @Override
104     public Client getClient() {
105         return client;
106     }
107
108     public void setClient(Client client) {
109         this.client = client;
110     }
111
112     @Override
113     public double getSize() {
114         return size;
115     }
116
117     public void setSize(double size) {
118         this.size = size;
119     }
120
121     public double getPercent() {
122         return percent;
123     }
124
125     public void setPercent(double percent) {
126         this.percent = percent;
127     }
128
129     public Date getTerm() {
130         return term;
131     }
132
133     public void setTerm(Date term) {
134         this.term = term;
135     }
136
137     public double getAccumulatedSum() {
138         return accumulatedSum;
139     }
140
141     public void setAccumulatedSum(double accumulatedSum) {
```

```
142             this.accumulatedSum = accumulatedSum;
143         }
144 
145         public boolean isStatus() {
146             return status;
147         }
148 
149         public void setStatus(boolean status) {
150             this.status = status;
151         }
152 }
```

Листинг 1.5: IAccount.java

```java
package ru.itmo.account;

import ru.itmo.bank.Transaction;
import ru.itmo.client.Client;
import ru.itmo.tools.BanksException;

import java.util.UUID;

public interface IAccount {
    UUID getId();

    Client getClient();

    double getSize();

    Transaction withdrawMoney(double money) throws BanksException;

    Transaction appendMoney(double money) throws BanksException;

    Transaction transferMoney(UUID accountId, double money) throws
    BanksException;

    double updateAccountSize(boolean check);
}
```

Листинг 1.6: Bank.java

```
1  package ru.itmo.bank;
2
3  import ru.itmo.account.IAccount;
4  import ru.itmo.client.Client;
5  import ru.itmo.tools.BanksException;
6
7  import java.util.ArrayList;
8  import java.util.List;
9  import java.util.UUID;
10
11 public class Bank {
12     private String name;
13     private double percent;
14     private double initialPercent;
15     private double commission;
16     private double maximumAvailableAmount;
17     private double limit;
18     private List<IAccount> accounts;
19     private List<Client> clients;
20     private List<Transaction> transactions;
21
22     public Bank(String name, double percent, double initialPercent,
       double commision,
23                 double maximumAvailableAmount, double limit) {
24         this.name = name;
25         this.percent = percent;
26         this.initialPercent = initialPercent;
27         this.commission = commision;
28         this.maximumAvailableAmount = maximumAvailableAmount;
29         this.limit = limit;
30         accounts = new ArrayList<IAccount>();
31         clients = new ArrayList<Client>();
32         transactions = new ArrayList<Transaction>();
33     }
34
35     public void changeBankPercent(double newPercent) {
36         percent = newPercent;
37     }
38
39     public void changeMaximumAvailableAmount(double newAmount) {
40         maximumAvailableAmount = newAmount;
41     }
42
43     public IAccount getAccountById(UUID id) throws BanksException {
44         for (IAccount account : accounts) {
45             if (account.getId() == id) {
46                 return account;
47             }
48         }
```

```
49          throw new BanksException("There is no account with that number
    .");
50      }
51
52      public String getName() {
53          return name;
54      }
55
56      public void setName(String name) {
57          this.name = name;
58      }
59
60      public double getPercent() {
61          return percent;
62      }
63
64      public void setPercent(double percent) {
65          this.percent = percent;
66      }
67
68      public double getInitialPercent() {
69          return initialPercent;
70      }
71
72      public void setInitialPercent(double initialPercent) {
73          this.initialPercent = initialPercent;
74      }
75
76      public double getCommission() {
77          return commission;
78      }
79
80      public void setCommission(double commission) {
81          this.commission = commission;
82      }
83
84      public double getMaximumAvailableAmount() {
85          return maximumAvailableAmount;
86      }
87
88      public void setMaximumAvailableAmount(double
    maximumAvailableAmount) {
89          this.maximumAvailableAmount = maximumAvailableAmount;
90      }
91
92      public double getLimit() {
93          return limit;
94      }
95
96      public void setLimit(double limit) {
```

```
97          this.limit = limit;
98      }
99
100     public List<IAccount> getAccounts() {
101         return accounts;
102     }
103
104     public void setAccounts(ArrayList<IAccount> accounts) {
105         this.accounts = accounts;
106     }
107
108     public List<Client> getClients() {
109         return clients;
110     }
111
112     public void setClients(ArrayList<Client> clients) {
113         this.clients = clients;
114     }
115
116     public List<Transaction> getTransactions() {
117         return transactions;
118     }
119
120     public void setTransactions(ArrayList<Transaction> transactions) {
121         this.transactions = transactions;
122     }
123 }
```

Листинг 1.7: CentralBank.java

```
1  package ru.itmo.bank;
2
3  import ru.itmo.account.CreditAccount;
4  import ru.itmo.account.DebitAccount;
5  import ru.itmo.account.DepositAccount;
6  import ru.itmo.account.IAccount;
7  import ru.itmo.client.Client;
8
9  import java.util.ArrayList;
10 import java.util.List;
11 import java.util.UUID;
12
13 public class CentralBank implements ICentralBank {
14     private List<Bank> banks = new ArrayList<Bank>();
15     private List<Client> clients = new ArrayList<Client>();
16     private List<Transaction> transactions = new ArrayList<Transaction
       >();
17
18     public List<Bank> getBanks() {
19         return banks;
20     }
21
22     public List<Client> getClients() {
23         return clients;
24     }
25
26     public Bank registerNewBank(Bank bank) {
27         banks.add(bank);
28         return bank;
29     }
30
31     public Client addNewClientToBank(Client client, Bank bank) {
32         clients.add(client);
33         bank.getClients().add(client);
34         return client;
35     }
36
37     public IAccount createNewAccount(Client client, Bank bank,
       TypeAccount typeAccount, double size) {
38         IAccount newAccount = null;
39         switch (typeAccount) {
40             case DEBIT:
41                 newAccount = new DebitAccount(bank, client, size);
42                 bank.getAccounts().add(newAccount);
43                 break;
44
45             // Deposit Account
46             case DEPOSIT:
47                 newAccount = new DepositAccount(bank, client, size);
```

```
48                  bank.getAccounts().add(newAccount);
49                  break;
50
51              // Credit Account
52              case CREDIT:
53                  newAccount = new CreditAccount(bank, client, size);
54                  bank.getAccounts().add(newAccount);
55                  break;
56          }
57          return newAccount;
58      }
59
60      public void notifyUpdateAccountSize(Bank bank) {
61          for (IAccount account : bank.getAccounts()) {
62              account.updateAccountSize(true);
63          }
64      }
65
66      public void canselTransaction(Bank bank, UUID transactionID) {
67          int index = -1;
68          for (Transaction transaction : transactions) {
69              if (transaction.getId() == transactionID) {
70                  index = transactions.indexOf(transaction);
71              }
72          }
73          transactions.remove(index);
74      }
75  }
```

Листинг 1.8: ICentralBank.java

```
1  package ru.itmo.bank;
2
3  import ru.itmo.account.IAccount;
4  import ru.itmo.client.Client;
5
6  import java.util.ArrayList;
7  import java.util.List;
8  import java.util.UUID;
9
10 public interface ICentralBank {
11     List<Bank> getBanks();
12
13     List<Client> getClients();
14
15     Bank registerNewBank(Bank bank);
16
17     Client addNewClientToBank(Client client, Bank bank);
18
19     IAccount createNewAccount(Client client, Bank bank, TypeAccount
       typeAccount, double accountSize);
20
21     void notifyUpdateAccountSize(Bank bank);
22
23     void canselTransaction(Bank bank, UUID transactionID);
24 }
```

Листинг 1.9: Transaction.java

```
1  package ru.itmo.bank;
2
3  import java.util.UUID;
4
5  public class Transaction {
6      private UUID id;
7      private UUID idAccount;
8      private UUID idRecipient;
9      private double amount;
10
11     public Transaction(UUID idAccount, double amount) {
12         id = UUID.randomUUID();
13         this.idAccount = idAccount;
14         this.amount = amount;
15         this.idRecipient = null;
16     }
17
18     public Transaction(UUID idAccount, UUID idRecipient, double amount
       ) {
19         id = UUID.randomUUID();
20         this.idAccount = idAccount;
21         this.idRecipient = idRecipient;
22         this.amount = amount;
23     }
24
25     public UUID getId() {
26         return id;
27     }
28
29     public UUID getIdAccount() {
30         return idAccount;
31     }
32
33     public void setId(UUID id) {
34         this.id = id;
35     }
36
37     public void setIdAccount(UUID idAccount) {
38         this.idAccount = idAccount;
39     }
40
41     public UUID getIdRecipient() {
42         return idRecipient;
43     }
44
45     public void setIdRecipient(UUID idRecipient) {
46         this.idRecipient = idRecipient;
47     }
48
```

```
49      public double getAmount() {
50          return amount;
51      }
52
53      public void setAmount(double amount) {
54          this.amount = amount;
55      }
56  }
```

Листинг 1.10: TypeAccount.java

```
1  package ru.itmo.bank;
2
3  import java.util.HashMap;
4  import java.util.Map;
5
6  public enum TypeAccount {
7      CREDIT("credit"),
8      DEBIT("debit"),
9      DEPOSIT("deposit");
10
11     private final String name;
12
13     TypeAccount(String name) {
14         this.name = name;
15     }
16
17     public String getName() {
18         return name;
19     }
20
21     private static final Map<String, TypeAccount> LOOKUP_MAP = new
       HashMap<>();
22
23     static {
24         for (TypeAccount env : values()) {
25             LOOKUP_MAP.put(env.getName(), env);
26         }
27     }
28
29     public static TypeAccount getTypeByName(String url) {
30         return LOOKUP_MAP.get(url);
31     }
32 }
```

Листинг 1.11: Client.java

```
1  package ru.itmo.client;
2
3  import java.util.UUID;
4
5  public class Client {
6      private String name;
7      private String surname;
8      private String address;
9      private String passport;
10
11     public Client(String clientName, String clientSurname) {
12         name = clientName;
13         surname = clientSurname;
14         address = "Default address";
15         passport = "Default passport";
16     }
17
18     public Client(String clientName, String clientSurname, String
       clientAddress, String clientPassport) {
19         name = clientName;
20         surname = clientSurname;
21         address = clientAddress;
22         passport = clientPassport;
23     }
24
25
26 }
```

Листинг 1.12: ConsoleInterface.java

```
1  package ru.itmo.service;
2
3  import ru.itmo.account.IAccount;
4  import ru.itmo.bank.Bank;
5  import ru.itmo.bank.ICentralBank;
6  import ru.itmo.bank.Transaction;
7  import ru.itmo.bank.TypeAccount;
8  import ru.itmo.client.Client;
9  import ru.itmo.tools.BanksException;
10
11 import java.util.*;
12
13 public class ConsoleInterface implements IConsoleInterface {
14     private ICentralBank centralBank;
15     private Bank bank;
16     private HashMap<String, String> dataBasePasswords = new HashMap
       <>();
17     private HashMap<String, Client> dataBaseClients = new HashMap<>();
18
19     public ConsoleInterface(ICentralBank centralBank, Bank bank) {
20         this.centralBank = centralBank;
21         this.bank = bank;
22     }
23
24     public Client greeting() {
25         Scanner scanner = new Scanner(System.in);
26         System.out.println("Welcome!");
27         System.out.println("Are you already registered in the system?
       Y/N");
28         String answer = scanner.nextLine();
29         if (answer.equals("Y")) {
30             System.out.println("Enter your username:");
31             String username = scanner.nextLine();
32             System.out.println("Enter your password:");
33             String password = scanner.nextLine();
34             if (dataBasePasswords.get(username) == password) {
35                 System.out.println("OK");
36             }
37             else {
38                 System.out.println("Invalid password");
39             }
40             return dataBaseClients.get(username);
41         } else {
42             return registration();
43         }
44     }
45
46     public Client registration() {
47         Scanner scanner = new Scanner(System.in);
```

```
48          System.out.println("Okey, then you can register right now.");
49          System.out.println("Enter your new username:");
50          String username = scanner.nextLine();
51          System.out.println("Enter your new password:");
52          String password = scanner.nextLine();
53          dataBasePasswords.put(username, password);
54          System.out.println("Enter your name:");
55          String name = scanner.nextLine();
56          System.out.println("Enter your surname:");
57          String surname = scanner.nextLine();
58          System.out.println("Do you want to enter an address? Y/N");
59          String answer = scanner.nextLine();
60          String address = (answer.equals("Y")) ? scanner.nextLine() :
    null;
61          System.out.println("Do you want to enter an passport? Y/N");
62          answer = scanner.nextLine();
63          String passport = (answer.equals("Y")) ? scanner.nextLine() :
    null;
64          var newClient = new Client(name, surname, address, passport);
65          centralBank.addNewClientToBank(newClient, bank);
66          return newClient;
67      }
68
69      public void createNewAccount(Client client) {
70          Scanner scanner = new Scanner(System.in);
71          System.out.println("What kind of account do you want to create
    ?");
72          System.out.println("1 - Debit\n2 - Deposit\n3 - Credit");
73          System.out.println("Write name of account:");
74          String nameAccount = scanner.nextLine().toLowerCase(Locale.
    ROOT);
75          TypeAccount typeAccount = TypeAccount.getTypeByName(
    nameAccount);
76          System.out.println("Write the amount of money you want to put
    into the account:");
77          double money = scanner.nextDouble();
78          centralBank.createNewAccount(client, bank, typeAccount, money)
    ;
79      }
80
81      public void workingWithAccount(Client client) throws
    BanksException {
82          Scanner scanner = new Scanner(System.in);
83          System.out.println("Do you already have an active account? Y/N
    ");
84          String answer = scanner.next();
85          while (!answer.equals("Y")) {
86              createNewAccount(client);
87              System.out.println("Do you already have an active account?
     Y/N");
```

```
88              answer = scanner.next();
89          }
90
91          ArrayList<IAccount> accounts = new ArrayList<IAccount>();
92          for (IAccount account : bank.getAccounts()) {
93              if (account.getClient() == client) {
94                  System.out.println(account.getId());
95                  accounts.add(account);
96              }
97          }
98
99          System.out.println("Choose necessary account and enter number.
    ");
100         int numAccount = scanner.nextInt() - 1;
101         UUID accountNumber = accounts.get(numAccount).getId();
102         int numOperation = 0;
103         while (numOperation != 5) {
104             System.out.println("What kind of operation do you want to
    do?");
105             System.out.println("1 - Withdraw money\n2 - Append money\
    n3 - Transfer money\n4 - Show the balance");
106             System.out.println("5 - Cansel transaction\n6 - Change
    account");
107             System.out.println("Write number:");
108             numOperation = scanner.nextInt();
109             double money;
110             switch (numOperation) {
111                 case 1 -> {
112                     System.out.println("Enter amount of money:");
113                     money = scanner.nextDouble();
114                     accounts.get(numAccount).withdrawMoney(money);
115                 }
116                 case 2 -> {
117                     System.out.println("Enter amount of money:");
118                     money = scanner.nextDouble();
119                     accounts.get(numAccount).appendMoney(money);
120                 }
121                 case 3 -> {
122                     System.out.println("Enter amount of money:");
123                     money = scanner.nextDouble();
124                     for (IAccount account : bank.getAccounts()) {
125                         System.out.println(account.getId());
126                     }
127                     System.out.println("Choose necessary account and
    enter number.");
128                     int num = scanner.nextInt();
129                     accounts.get(numAccount).transferMoney(accounts.
    get(num).getId(), money);
130                 }
131                 case 4 -> {
```

```
132                     System.out.println(accounts.get(numAccount).
    getSize());
133                 }
134                 case 5 -> {
135                     ArrayList<Transaction> transactions = new
    ArrayList<Transaction>();
136                     for (Transaction transaction : bank.
    getTransactions()) {
137                         if (transaction.getIdAccount() == accounts.get
    (numAccount).getId()) {
138                             System.out.println(transaction.getId());
139                             transactions.add(transaction);
140                         }
141                     }
142                     System.out.println("Choose necessary transaction
    and enter number.");
143                     int numTransaction = scanner.nextInt() - 1;
144                     Transaction currentTransaction = transactions.get(
    numTransaction);
145                     if (currentTransaction.getIdRecipient() == null) {
146                         bank.getAccountById(currentTransaction.
    getIdAccount()).
147                                 appendMoney(currentTransaction.
    getAmount());
148                         bank.getTransactions().remove(bank.
    getTransactions().indexOf(currentTransaction));
149                     } else {
150                         bank.getAccountById(currentTransaction.
    getIdAccount()).
151                                 appendMoney(currentTransaction.
    getAmount());
152                         bank.getAccountById(currentTransaction.
    getIdRecipient()).
153                                 withdrawMoney(currentTransaction.
    getAmount());
154                         bank.getTransactions().remove(bank.
    getTransactions().indexOf(currentTransaction));
155                     }
156                 }
157             }
158         }
159     }
160 }
```

Листинг 1.13: IConsoleInterface.java

```java
package ru.itmo.service;

import ru.itmo.client.Client;
import ru.itmo.tools.BanksException;

public interface IConsoleInterface {
    Client greeting();

    Client registration();

    void workingWithAccount(Client client) throws BanksException;

    void createNewAccount(Client client);
}
```

Листинг 1.14: BankTest.java

```
1  package ru.itmo.tests;
2
3  import org.junit.Assert;
4  import org.testng.annotations.BeforeTest;
5  import org.testng.annotations.Test;
6  import ru.itmo.account.IAccount;
7  import ru.itmo.bank.Bank;
8  import ru.itmo.bank.CentralBank;
9  import ru.itmo.bank.ICentralBank;
10 import ru.itmo.bank.TypeAccount;
11 import ru.itmo.client.Client;
12 import ru.itmo.tools.BanksException;
13
14 import static java.lang.Math.abs;
15
16 public class BankTest {
17     private ICentralBank centralBank;
18     double epsilon = 0.00000001;
19
20     @BeforeTest
21     public void setUp() {
22         centralBank = new CentralBank();
23     }
24
25     @Test
26     public void createNewBank() {
27         Bank newBank = new Bank("Sberbank", 13.1, 3, 15000,
28                 5000, 100000);
29         centralBank.registerNewBank(newBank);
30         Assert.assertNotEquals(centralBank.getBanks().indexOf(newBank)
    , -1);
31     }
32
33     @Test
34     public void createNewClient_addClientToBank() {
35         Bank bank = new Bank("Sberbank", 13.1, 3, 15000,
36                 5000, 100000);
37         centralBank.registerNewBank(bank);
38         Client newClient = new Client("Sergey", "Potapov");
39         centralBank.addNewClientToBank(newClient, bank);
40         Assert.assertNotEquals(bank.getClients().indexOf(newClient),
    -1);
41     }
42
43     @Test
44     public void createNewDebitAccount_doTransactions() throws
    BanksException {
45         Bank bank = new Bank("Sberbank", 13.1, 3, 15000,
46                 5000, 100000);
```

```
47          centralBank.registerNewBank(bank);
48          Client newClient = new Client("Sergey", "Potapov");
49          centralBank.addNewClientToBank(newClient, bank);
50          IAccount newAccount = centralBank.createNewAccount(newClient,
    bank, TypeAccount.DEBIT, 40000);
51          newAccount.appendMoney(1000);
52          newAccount.withdrawMoney(200);
53          Assert.assertTrue(abs(newAccount.getSize() - (40000 + 1000 -
    200)) < epsilon);
54          //Assert.assertEquals(newAccount.getSize(), 40000 + 1000 -
    200);
55      }
56
57      @Test(expectedExceptions = BanksException.class)
58      public void createNewAccount_notEnoughMoney_throwException()
    throws BanksException {
59          Bank bank = new Bank("Sberbank", 13.1, 3, 15000,
60                  5000, 100000);
61          centralBank.registerNewBank(bank);
62          Client newClient = new Client("Sergey", "Potapov");
63          centralBank.addNewClientToBank(newClient, bank);
64          IAccount newAccount = centralBank.createNewAccount(newClient,
    bank, TypeAccount.DEBIT, 1000);
65          newAccount.withdrawMoney(2000);
66      }
67
68      @Test(expectedExceptions = BanksException.class)
69      public void createNewDoubtfulAccount_throwException() throws
    BanksException {
70          Bank bank = new Bank("Sberbank", 13.1, 3, 15000,
71                  5000, 100000);
72          centralBank.registerNewBank(bank);
73          Client newClient = new Client("Sergey", "Potapov");
74          centralBank.addNewClientToBank(newClient, bank);
75          IAccount newAccount = centralBank.createNewAccount(newClient,
    bank, TypeAccount.DEBIT, 40000);
76          newAccount.withdrawMoney(6000);
77      }
78  }
```

Листинг 1.15: BanksException.java

```
1 package ru.itmo.tools;
2
3 public class BanksException extends Exception {
4     public BanksException(String message) {
5         super(message);
6     }
7 }
```