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:

Колпикова Ксения Денисовна
группа: M32071

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

```
import models.Bank;
import models.CentralBank;
import models.Client;
import models.PercentOfDeposit;
import services.ClientBuilder;
import tools.BanksException;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) throws BanksException {
        CentralBank central = null;
        Scanner in = new Scanner(System.in);
        System.out.println("1 – РЎРѕР·РґР°РЅРёРµ РїРѕР»СЊР·РѕРІР°
    РєР»РёРµРЅС‚Р° ");
        System.out.println("2 – РЎРѕР·РґР°РЅРёРµ РєР»РёРµРЅС‚Р° СЃ
    РѕРїС†РёРѕРЅР°Р»СЊРЅС‹РјРё");
        System.out.println("Name, Surname, Address, Passport");
        String str = in.nextLine();
        ClientBuilder creation = null;
        Client client = null;
        Bank bank = null;
        switch (str)
        {
            case "1":
                client = creation.changeName(in.nextLine()).
                        changeSurname(in.nextLine()).changeAddress(in.
    nextLine()).
                        changePassport(in.nextLine()).create();
                break;
            case "2":
                client = creation.changeName(in.nextLine()).
    changeSurname(in.nextLine()).create();
                break;
        }

        while (str != "exit")
        {
            System.out.println("1 – РЎРѕР·РґР°РЅРёРµ Р±Р°РЅРєР°");
            System.out.println("2 – Р"РѕР±Р°РІР»РµРЅРёРµ РєР»РёРµРЅС‚Р°
    РІ Р±Р°РЅРє");
            System.out.println("3 – РЎРѕР·РґР°РЅРёРµ СЃС‡РµС‚Р° РІ
    Р±Р°РЅРєРµ");
            System.out.println("4 – РЎРЅСЏС‚РёРµ РґРµРїРѕР·РёС‚Р° СЃРѕ
    СЃС‡РµС‚Р°");
            System.out.println("5 – РџРѕРїРѕР»РЅРµРЅРёРµ СЃС‡РµС‚Р°");
            System.out.println("6 – РџРµСЂРµРІРѕРґ РґРµРЅРµРі");
            System.out.println("7 – РћС‚РјРµРЅР° РїРѕРїРѕР»РЅРµРЅРёРЯ");
```

```
42              System.out.println("8 – РћС,РјРµРSР° СЃРSСЏС,РёСЏ");
43              System.out.println("9 – РћС,РјРµРSР° РїРµСЂРµРІРsРґР°");
44              System.out.println("10 – РџСЂРsРjРsС,РєР° РІСЂРµРjРµРSРё");
45              System.out.println("exit – program finish");
46              str = in.nextLine();
47              switch (str)
48              {
49                  case "1":
50                      PercentOfDeposit percent = null;
51                      for (int j = 0; j < 2; j++)
52                      {
53                          System.out.println("Р'РІРµРґРёС,Рµ СЃСумРjРjСґ РёРїСЂРsС†РµРSС,");
54                          percent.addPercentAndSum(in.nextInt(), in.nextDouble());
55                      }
56                      System.out.println("Р'РІРµРґРёС,Рµ РїР°СЂР°РjРµС,СЂС<Р±Р°РSРєР°");
57
58                      bank = new Bank(in.nextLine(), in.nextDouble(), in.nextDouble(), in.nextDouble(), in.nextDouble(), percent);
59                      central.registerBank(bank);
60                      break;
61                  case "2":
62                      bank.addClientInBank(client);
63                      break;
64                  case "3":
65                      switch (in.nextLine())
66                      {
67                          case "1":
68                              System.out.println("Р'РІРµРґРёС,Рµ СЃСумРjРjСґ РёСЃСЂРsРє");
69                              central.creationAccount(bank, in.nextInt(), client, in.nextInt(), "Credit");
70                              break;
71                          case "2":
72                              System.out.println("Р'РІРµРґРёС,Рµ СЃСумРjРjСґ РёСЃСЂРsРє");
73                              central.creationAccount(bank, in.nextInt(), client, in.nextInt(), "Debit");
74                              break;
75                          case "3":
76                              System.out.println("Р'РІРµРґРёС,Рµ СЃСумРjРjСґ РёСЃСЂРsРє");
77                              central.creationAccount(bank, in.nextInt(), client, in.nextInt(), "Deposit");
78                              break;
79                          default:
80                              throw new BanksException("Invalid option");
```

```
81                      }
82
83                      break;
84                  case "4":
85                      System.out.println("P'PIPµPrPëC,Pµ CЃCґPjPjCґ Pë
    PSPsPjPµCЂ CЃC‡PµC,P°");
86                      central.withdrawalMoney(bank, in.nextInt(), in.
    nextInt(), client);
87                      break;
88                  case "5":
89                      System.out.println("P'PIPµPrPëC,Pµ CЃCґPjPjCґ Pë
    PSPsPjPµCЂ CЃC‡PµC,P°");
90                      central.refillMoneyOn(bank, in.nextInt(), in.
    nextInt());
91                      break;
92                  case "6":
93                      System.out.println("P'PIPµPrPëC,Pµ CЃCґPjPjCґ,
    PSPsPjPµCЂ CЃC‡PµC,P° PsC,PїCЂP°PIPëC,PµP»CЏ Pë PSPsPjPµCЂ CЃC‡PµC,P°
    PїPsP»CґC‡P°C,PµP»CЏ");
94                      central.transferMoneyOnBalance(bank, in.nextInt(),
     in.nextInt(), bank, in.nextInt(), client);
95                      break;
96                  case "7":
97                      System.out.println("P'PIPµPrPëC,Pµ CЃCґPjPjCґ Pë
    PSPsPjPµCЂ CЃC‡PµC,P° ");
98                      central.cancelRefill(bank, in.nextInt(), in.
    nextInt());
99                      break;
100                 case "8":
101                     System.out.println("P'PIPµPrPëC,Pµ CЃCґPjPjCґ Pë
    PSPsPjPµCЂ CЃC‡PµC,P° ");
102                     central.cancelWithdrawal(bank, in.nextInt(), in.
    nextInt());
103                     break;
104                 case "9":
105                     System.out.println("P'PIPµPrPëC,Pµ CЃCґPjPjCґ,
    PSPsPjPµCЂ CЃC‡PµC,P° PsC,PїCЂP°PIPëC,PµP»CЏ Pë PSPsPjPµCЂ CЃC‡PµC,P°
    PїPsP»CґC‡P°C,PµP»CЏ");
106                     central.cancelTransfer(bank, in.nextInt(), in.
    nextInt(), bank, in.nextInt());
107                     break;
108                 case "10":
109                     System.out.println("P'PIPµPrPëC,Pµ CЃCґPjPjCґ Pë
    PSPsPjPµCЂ CЃC‡PµC,P° ");
110                     central.checkTime( in.nextInt(), bank.getAccountId
    ( in.nextInt()));
111                     break;
112
113                 case "exit":
114                     System.out.println("finish");
```

```
115                         break;
116                     }
117                 }
118             }
119 }
```

Листинг 1.2: Bank.java

```
1  package models;
2
3  import services.EventManager;
4  import services.IAccount;
5  import tools.BanksException;
6
7  import java.util.ArrayList;
8  import java.util.List;
9
10 public class Bank {
11     public EventManager eventManager;
12     private List<Client> clients;
13     private List<IAccount> accountsOfClient;
14     public PercentOfDeposit PercentsOfDeposit;
15     public String nameOfBank;
16     public double commissionOfCreditOfBank;
17     public double limitOfCreditOfBank;
18     public double percentOfDebitOfBank;
19     public double limitationOfBank;
20
21     public Bank(String name, double commissionOfCredit, double
    limitOfCredit, double percentOfDebit, double limitation,
22                 PercentOfDeposit percentOfDeposit) throws
    BanksException {
23         if (percentOfDeposit == null) throw new BanksException("
    Invalid percents of deposit");
24         PercentsOfDeposit = percentOfDeposit;
25         if (name == null) throw new BanksException("Invalid name");
26         nameOfBank = name;
27         commissionOfCreditOfBank = commissionOfCredit;
28         limitOfCreditOfBank = limitOfCredit;
29         percentOfDebitOfBank = percentOfDebit;
30         clients = new ArrayList<>();
31         limitationOfBank = limitation;
32         accountsOfClient = new ArrayList<>();
33         eventManager = new EventManager("Change nameOfBank",
34                 "Change commissionOfCreditOfBank", "Change
    commissionOfCreditOfBank",
35                 "Change limitOfCreditOfBank", "Change
    percentOfDebitOfBank", "Change limitationOfBank");
36     }
37
38     public void addClientInBank(Client client) throws BanksException {
39         if (client == null) throw new BanksException("Invalid client")
    ;
40         if (clients.stream().anyMatch(n -> n.nameOfClient == client.
    nameOfClient) &&
41                 clients.stream().anyMatch(n -> n.surnameOfBank ==
    client.surnameOfBank))
```

```
42              throw new BanksException("Client already in use");
43          clients.add(client);
44      }
45
46      public void setNameOfBank(String name) {
47           this.nameOfBank = name;
48           eventManager.notify("Change nameOfBank");
49      }
50      public String getNameOfBank() {
51          return this.nameOfBank;
52      }
53
54      public void setCommissionOfCredit(double commissionOfCredit)
55      {
56          this.commissionOfCreditOfBank = commissionOfCredit;
57          eventManager.notify("Change commissionOfCreditOfBank");
58      }
59      public double getCommissionOfCredit()
60      {
61          return this.commissionOfCreditOfBank;
62      }
63
64      public void setLimitOfCredit(double limitOfCredit)
65      {
66          this.limitOfCreditOfBank = limitOfCredit;
67          eventManager.notify("Change limitOfCreditOfBank");
68      }
69      public double getLimitOfCredit()
70      {
71          return this.limitOfCreditOfBank;
72      }
73
74      public void setPercentOfDebit(double percentOfDebit)
75      {
76          this.percentOfDebitOfBank = percentOfDebit;
77          eventManager.notify("Change percentOfDebitOfBank");
78      }
79      public double getPercentOfDebit()
80      {
81          return this.percentOfDebitOfBank;
82      }
83
84      public void setLimitation(double limitation)
85      {
86          this.limitationOfBank = limitation;
87          eventManager.notify("Change limitationOfBank");
88      }
89      public double getLimitation()
90      {
91          return this.limitationOfBank;
```

```
92      }
93
94      public void addDepositInIAccount(Deposit deposit) throws
    BanksException {
95          if (deposit == null) throw new BanksException("Invalid deposit
    ");
96          accountsOfClient.add(deposit);
97      }
98
99      public void addDebitInIAccount(Debit debit) throws BanksException
    {
100         if (debit == null) throw new BanksException("Invalid debit");
101         accountsOfClient.add(debit);
102     }
103
104     public void addCreditInIAccount(Credit credit) throws
    BanksException {
105         if (credit == null) throw new BanksException("Invalid credit")
    ;
106         accountsOfClient.add(credit);
107     }
108
109     public boolean isOperationInAccount(int sum) {
110         return accountsOfClient.stream().allMatch(x -> x.withdraw(sum)
    );
111     }
112
113     public IAccount getAccountId(int id) {
114         IAccount account = null;
115         for (int i = 0; i < accountsOfClient.size(); i++) {
116             if (accountsOfClient.remove(i).getAccountId() == id)
    account = accountsOfClient.remove(i);
117         }
118         return account;
119     }
120
121     public int countClients() {
122         return clients.size();
123     }
124
125     public int countAccounts() {
126         return accountsOfClient.size();
127     }
128 }
```

Листинг 1.3: CentralBank.java

```
1  package models;
2
3  import services.IAccount;
4  import tools.BanksException;
5
6  import java.util.ArrayList;
7  import java.util.List;
8
9  public class CentralBank
10 {
11     private static int countOfId = 0;
12     private List<Bank> banks;
13
14     public CentralBank() {
15
16         banks = new ArrayList<>();
17     }
18
19     public void registerBank(Bank bank) throws BanksException {
20         if (bank == null) throw new BanksException("Invalid Bank");
21         banks.add(bank);
22     }
23
24     public IAccount creationAccount(Bank bank, int balance, Client
    client, int time, String option) throws BanksException {
25         if (option == null) throw new BanksException("Invalid option")
    ;
26         if (client == null) throw new BanksException("Invalid client")
    ;
27         if (bank == null) throw new BanksException("Invalid Bank");
28         if (banks.stream().anyMatch(b -> b == bank)) return
    creationAccountForClient(bank, client, balance, time, option);
29         else throw new BanksException("Invalid bank");
30     }
31
32     public void checkTime(int time, IAccount account) throws
    BanksException {
33         if (account == null) throw new BanksException("Invalid account
    ");
34         account.benefitPay(time);
35     }
36
37     public IAccount creationAccountForClient(Bank bank, Client client,
     int balance, int time, String option) throws BanksException {
38         switch (option)
39         {
40             case "Credit":
41
42                 Credit credit = new Credit(countOfId++, bank.
```

```
    commissionOfCreditOfBank, bank.limitOfCreditOfBank, balance);
43                  bank.addCreditInlAccount(credit);
44                  return credit;
45
46              case "Debit":
47
48                  Debit debit = new Debit(balance, bank.
    percentOfDebitOfBank, countOfId++);
49                  bank.addDebitInlAccount(debit);
50                  return debit;
51
52              case "Deposit":
53
54                  double depositPercent = 0;
55                  int f = 0;
56                  for ( Integer key : bank.PercentsOfDeposit.percents.
    keySet() )  {
57                  f++;
58                  if (balance < key) {
59                      depositPercent = bank.PercentsOfDeposit.percents.
    get(key);
60                  }
61                  if( bank.PercentsOfDeposit.percents.keySet().size() -
    f == 1 && depositPercent == 0)
62                      depositPercent = bank.PercentsOfDeposit.percents.
    get(key);
63              }
64                  Deposit deposit = new Deposit(countOfId++, balance,
    time, depositPercent);
65              bank.addDepositInlAccount(deposit);
66              return deposit;
67              default:
68                  throw new BanksException("Option not found");
69          }
70      }
71
72      public void refillMoneyOn(Bank bank, int balance, int id) throws
    BanksException {
73          if (bank == null) throw new BanksException("Invalid Bank");
74          bank.getAccountId(id).refillOperation(balance);
75      }
76
77      public void withdrawalMoney(Bank bank, int balance, int id, Client
        client) throws BanksException {
78          if (!checkClientPassportAndAddress(client) && balance > bank.
    limitationOfBank && bank.isOperationInAccount(balance)) {
79              throw new BanksException("Invalid Withdrawal");
80          }
81          bank.getAccountId(id).withdrawalOperation(balance);
82      }
```

```
83
84      public void transferMoneyOnBalance(Bank bank1, int balance, int
    id1, Bank bank2, int id2, Client client) throws BanksException {
85          if (!checkClientPassportAndAddress(client) && balance > bank1.
    limitationOfBank && bank1.isOperationInAccount(balance)) {
86              throw new BanksException("Invalid Transfer");
87          }
88          bank1.getAccountId(id1).transferOperation(bank2.getAccountId(
    id2), balance);
89      }
90
91      public void cancelRefill(Bank bank, int balance, int id) throws
    BanksException {
92          IAccount operation = bank.getAccountId(id);
93          if (operation == null) throw new BanksException("Refill don't
    exists");
94          operation.cancelRefillOperation(balance);
95      }
96
97      public void cancelTransfer(Bank bank1, int balance, int id1, Bank
    bank2, int id2) throws BanksException {
98          IAccount operation = bank1.getAccountId(id1);
99          if (operation == null) throw new BanksException("Transfer don'
    t exists");
100         operation.cancelTransferOperation(bank2.getAccountId(id2),
    balance);
101     }
102
103     public void cancelWithdrawal(Bank bank, int balance, int id)
    throws BanksException {
104         IAccount operation = bank.getAccountId(id);
105         if (operation == null) throw new BanksException("Withdrawal
    don't exists");
106         operation.cancelWithdrawalOperation(balance);
107     }
108
109     private boolean checkClientPassportAndAddress(Client client) {
110         return !client.passportOfBank.isBlank() || !client.
    passportOfBank.isBlank();
111     }
112     public int countBanks(){
113         return banks.size();
114     }
115 }
```

Листинг 1.4: Client.java

```
package models;

import tools.BanksException;

public class Client {
    private static int _id = 0;
    public int Id = 0;
    public String nameOfClient;
    public String surnameOfBank;
    public String addressOfBank;
    public String passportOfBank;

    public Client(String name, String surname, String address, String
   passport) throws BanksException {
        Id = _id++;
        if(name == null) {
            throw new BanksException("Invalid name");
        }
        nameOfClient = name;
        if(surname == null) {
            throw new BanksException("Invalid surname");
        }
        surnameOfBank = surname;
        addressOfBank = address;
        passportOfBank = passport;
    }
    public static void update(String event){
        System.out.print(event);
    }
}
```

Листинг 1.5: Credit.java

```
1  package models;
2
3  import services.IAccount;
4
5  public class Credit implements IAccount {
6      private double commissionTmp = 0;
7      public double commissionOfCredit;
8      public double limitOfBank;
9      public int idOfBank;
10     public int balanceOfBank;
11
12     public Credit(int id, double commission, double limit, int balance
       )
13     {
14         commissionOfCredit = commission;
15         limitOfBank = limit;
16         idOfBank = id;
17         balanceOfBank = balance;
18     }
19
20     public void withdrawalOperation(int sum) {
21
22         balanceOfBank -=sum;
23     }
24
25     public void cancelWithdrawalOperation(int sum) {
26
27         balanceOfBank +=sum;
28     }
29
30     public void refillOperation(int sum) {
31
32         balanceOfBank +=sum;
33     }
34
35     public void cancelRefillOperation(int sum) {
36
37         balanceOfBank -=sum;
38     }
39
40     public void transferOperation(IAccount other, int sum) {
41         balanceOfBank -=sum;
42         other.refillOperation(sum);
43     }
44
45     public void cancelTransferOperation(IAccount other, int sum) {
46         balanceOfBank +=sum;
47         other.withdrawalOperation(sum);
48     }
```

```
49
50      public void benefitPay(int time) {
51          commissionTmp += balanceOfBank * commissionOfCredit /300;
52          balanceOfBank -=(int) commissionTmp *time;
53          commissionTmp =0;
54      }
55
56      public boolean withdraw(int sum) {
57          return Math.abs(balanceOfBank -sum)< limitOfBank;
58      }
59
60      public int getAccountId(){
61
62          return idOfBank;
63      }
64
65      public int checkBalance() {
66
67          return balanceOfBank;
68      }
69  }
```

Листинг 1.6: Debit.java

```
1  package models;
2
3  import services.IAccount;
4
5  public class Debit implements IAccount {
6      private double benefit = 0;
7      public double Percent;
8      public int Balance;
9      public int Id;
10
11     public Debit(int balance, double percent, int id) {
12         Percent = percent;
13         Balance = balance;
14         Id = id;
15     }
16
17
18     public void withdrawalOperation(int sum) {
19
20         Balance -= sum;
21     }
22
23     public void cancelWithdrawalOperation(int sum) {
24
25         Balance += sum;
26     }
27
28     public void refillOperation(int sum) {
29
30         Balance += sum;
31     }
32
33     public void cancelRefillOperation(int sum) {
34
35         Balance -= sum;
36     }
37
38     public void transferOperation(IAccount account2, int sum) {
39         Balance -= sum;
40         account2.refillOperation(sum);
41     }
42
43     public void cancelTransferOperation(IAccount other, int sum) {
44         Balance += sum;
45         other.withdrawalOperation(sum);
46     }
47
48     public void benefitPay(int time) {
49         benefit = Balance * Percent / 300;
```

```
50          Balance += (int) benefit * time;
51          benefit = 0;
52      }
53
54      public boolean withdraw(int sum) {
55
56          return Balance >= sum;
57      }
58
59      public int getAccountId() {
60
61          return Id;
62      }
63
64      public int checkBalance() {
65
66          return Balance;
67      }
68  }
```

Листинг 1.7: Deposit.java

```
1  package models;
2
3
4  import services.IAccount;
5
6  public class Deposit implements IAccount {
7      private double benefit = 0;
8      public int idOfDeposit;
9      public int timeOfDeposit;
10     public double percentOfDeposit;
11     public int balanceOfDeposit;
12
13     public Deposit(int id, int balance, int time, double percent) {
14         idOfDeposit = id;
15         timeOfDeposit = time;
16         percentOfDeposit = percent;
17         balanceOfDeposit = balance;
18     }
19
20
21     public void withdrawalOperation(int sum) {
22
23         balanceOfDeposit -= sum;
24     }
25
26     public void cancelWithdrawalOperation(int sum) {
27
28         balanceOfDeposit += sum;
29     }
30
31     public void refillOperation(int sum) {
32
33         balanceOfDeposit += sum;
34     }
35
36     public void cancelRefillOperation(int sum) {
37
38         balanceOfDeposit -= sum;
39     }
40
41     public void transferOperation(IAccount other, int sum) {
42         balanceOfDeposit -= sum;
43         other.refillOperation(sum);
44     }
45
46     public void cancelTransferOperation(IAccount other, int sum) {
47         balanceOfDeposit += sum;
48         other.withdrawalOperation(sum);
49     }
```

```
50
51      public void benefitPay(int time) {
52          benefit = balanceOfDeposit * percentOfDeposit / 300;
53          balanceOfDeposit += (int) benefit * time;
54          benefit = 0;
55      }
56
57      public boolean withdraw(int sum) {
58
59          return balanceOfDeposit >= sum && timeOfDeposit == 0;
60      }
61
62      public int getAccountId() {
63
64          return idOfDeposit;
65      }
66
67      public int checkBalance() {
68
69          return balanceOfDeposit;
70      }
71  }
```

Листинг 1.8: PercentOfDeposit.java

```
1  package models;
2
3  import java.util.*;
4
5  public class PercentOfDeposit
6  {
7      public Map<Integer, Double> percents;
8      public PercentOfDeposit() {
9
10         percents = new HashMap<>();
11     }
12
13     public void addPercentAndSum(int sum, double percent)
14     {
15         percents.put(sum,percent);
16     }
17 }
```

Листинг 1.9: ClientBuilder.java

```
package services;

import models.Client;
import tools.BanksException;

public class ClientBuilder
{
    private String name;
    private String passport;
    private String surname;
    private String address;

    public ClientBuilder changeName(String name) throws BanksException
    {
        if(name == null) throw new BanksException("Invalid name");
        this.name = name;
        return this;
    }

    public ClientBuilder changeSurname(String surname) throws
    BanksException {
        if(surname == null) throw new BanksException("Invalid surname"
    );
        this.surname = surname;
        return this;
    }

    public ClientBuilder changePassport(String passport) throws
    BanksException {
        if(passport == null) throw new BanksException("Invalid
    passport");
        this.passport = passport;
        return this;
    }

    public ClientBuilder changeAddress(String address) throws
    BanksException {
        if(address == null) throw new BanksException("Invalid address"
    );
        this.address = address;
        return this;
    }

    public Client create() throws BanksException {
        return new Client(name, surname, address, passport);
    }
}
```

Листинг 1.10: EventManager.java

```
1  package services;
2
3
4  import models.Client;
5
6  import java.util.*;
7
8  public class EventManager {
9      Map<String, List<Client>> listeners = new HashMap<>();
10
11     public EventManager(String... operations) {
12         for (String operation : operations) {
13             this.listeners.put(operation, new ArrayList<>());
14         }
15     }
16
17     public void subscribe(String eventType, Client listener) {
18         List<Client> users = listeners.get(eventType);
19         users.add(listener);
20     }
21
22     public void unsubscribe(String eventType, Client listener) {
23         List<Client> users = listeners.get(eventType);
24         users.remove(listener);
25     }
26
27     public void notify(String event) {
28         List<Client> users = listeners.get(event);
29         for (Client listener : users) {
30             listener.update(event);
31         }
32     }
33 }
```

Листинг 1.11: IAccount.java

```
1  package services;
2
3  public interface IAccount
4  {
5      public void withdrawalOperation(int sum);
6      public void cancelWithdrawalOperation(int sum);
7      public void refillOperation(int sum);
8      public void cancelRefillOperation(int sum);
9      public void transferOperation(IAccount other, int sum);
10     public void cancelTransferOperation(IAccount other, int sum);
11     public void benefitPay(int time);
12     public boolean withdraw(int sum);
13     public int getAccountId();
14     public int checkBalance();
15 }
```

Листинг 1.12: BanksException.java

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