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:

Жарков Григорий Алексеевич
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: Console.java

```
1  import account.AccountType;
2  import account.DepositConsole;
3  import account.DepositDTO;
4  import account.IAccount;
5  import bank.Bank;
6  import bank.BankConsole;
7  import bank.BankDTO;
8  import bank.CentralBank;
9  import client.Client;
10 import client.ClientConsole;
11 import client.ClientDTO;
12 import tools.BankException;
13
14 import java.util.Scanner;
15
16 public class Console {
17     private CentralBank centralBank;
18
19     public Console(CentralBank centralBank) throws BankException {
20         if (centralBank == null) {
21             var e = new IllegalArgumentException();
22             throw new BankException("Central bank can not be null!", e
       );
23         }
24
25         this.centralBank = centralBank;
26     }
27
28     public void work() throws BankException {
29         while (true) {
30             System.out.println("Enter what you want to do: " +
31                     "\n1) Add new bank " +
32                     "\n2) Add new client " +
33                     "\n3) Add new account " +
34                     "\n4) Add money to account " +
35                     "\n5) Take money from account " +
36                     "\n6) Make transaction " +
37                     "\n0) Exit");
38
39             Scanner choice = new Scanner(System.in);
40             String answer = choice.nextLine();
41
42             switch (answer) {
43                 case "1" -> {
44                     BankDTO bankData = new BankConsole().
       collectBankData();
45                     centralBank.addBank(bankData);
46                 }
47                 case "2" -> {
```

```
48                  System.out.println("Choose in what bank you want
    to register new client:");
49                  for (Bank bank : centralBank.getBanks()) {
50                      System.out.println(bank.getName() + " ");
51                  }
52                  System.out.println();
53                  Scanner inCase2 = new Scanner(System.in);
54                  int bankCase2 = inCase2.nextInt();
55                  System.out.println("Now enter client data:");
56                  ClientDTO clientData = new ClientConsole().
    collectPersonalData();
57                  centralBank.getBanks().get(bankCase2).
    registerClient(clientData);
58              }
59              case "3" -> {
60                  System.out.println("Choose in what bank client
    registered:");
61                  for (Bank bank : centralBank.getBanks()) {
62                      System.out.println(bank.getName() + " ");
63                  }
64                  System.out.println();
65                  Scanner inCase3 = new Scanner(System.in);
66                  int bankCase3 = inCase3.nextInt();
67                  System.out.println("Choose who wants to open new
    account:");
68                  for (Client client : centralBank.getBanks().get(
    bankCase3).getClients()) {
69                      System.out.println(client.getId() + " ");
70                  }
71                  System.out.println();
72                  int clientCase3 = inCase3.nextInt();
73                  System.out.println("Choose what account type you
    want to create:" +
74                          " 1 for debit, " +
75                          "2 for deposit, " +
76                          "3 for credit");
77                  int accountCase3 = inCase3.nextInt();
78                  switch (accountCase3) {
79                      case 1:
80                          centralBank.getBanks().get(bankCase3).
    registerAccount(
81                                  centralBank.getBanks().get(
    bankCase3).getClients().get(clientCase3),
82                                  AccountType.Debit, null);
83                          break;
84                      case 2:
85                          System.out.println("Enter extra info:");
86                          DepositDTO depositData = new
    DepositConsole().collectDepositConditions();
```

```
87                              centralBank.getBanks().get(bankCase3).
    registerAccount(
88                                  centralBank.getBanks().get(
    bankCase3).getClients().get(clientCase3),
89                                  AccountType.Deposit, depositData);
90                              break;
91                          case 3:
92                              centralBank.getBanks().get(bankCase3).
    registerAccount(
93                                  centralBank.getBanks().get(
    bankCase3).getClients().get(clientCase3),
94                                  AccountType.Credit, null);
95                          default:
96                              var e = new IllegalArgumentException();
97                              throw new BankException("Invalid account
    type!", e);
98                      }
99                  }
100                 case "4" -> {
101                     System.out.println("Choose in what bank client
    registered:");
102                     for (Bank bank : centralBank.getBanks()) {
103                         System.out.println(bank.getName() + " ");
104                     }
105                     System.out.println();
106                     Scanner inCase4 = new Scanner(System.in);
107                     int bankCase4 = inCase4.nextInt();
108                     System.out.println("Choose account's owner");
109                     for (Client client : centralBank.getBanks().get(
    bankCase4).getClients()) {
110                         System.out.println(client.getId() + " ");
111                     }
112                     System.out.println();
113                     int clientCase4 = inCase4.nextInt();
114                     System.out.println("Choose account:");
115                     for (IAccount account :
116                             centralBank.getBanks().get(bankCase4).
    getClients().get(clientCase4).getAccounts()) {
117                         System.out.println(account.getId() + " ");
118                     }
119                     System.out.println();
120                     int accountCase4 = inCase4.nextInt();
121                     System.out.println("Enter amount:");
122                     double amountCase4 = inCase4.nextDouble();
123                     centralBank.getBanks().get(bankCase4).getClients()
    .get(clientCase4).
124                             getAccounts().get(accountCase4).addMoney(
    amountCase4);
125                 }
126                 case "5" -> {
```

```
127                     System.out.println("Choose in what bank client
    registered:");
128                     for (Bank bank : centralBank.getBanks()) {
129                         System.out.println(bank.getName() + " ");
130                     }
131                     System.out.println();
132                     Scanner inCase5 = new Scanner(System.in);
133                     int bankCase5 = inCase5.nextInt();
134                     System.out.println("Choose account's owner");
135                     for (Client client : centralBank.getBanks().get(
    bankCase5).getClients()) {
136                         System.out.println(client.getId() + " ");
137                     }
138                     System.out.println();
139                     int clientCase5 = inCase5.nextInt();
140                     System.out.println("Choose account:");
141                     for (IAccount account :
142                             centralBank.getBanks().get(bankCase5).
    getClients().get(clientCase5).getAccounts()) {
143                         System.out.println(account.getId() + " ");
144                     }
145                     System.out.println();
146                     int accountCase5 = inCase5.nextInt();
147                     System.out.println("Enter amount:");
148                     double amountCase5 = inCase5.nextDouble();
149                     centralBank.getBanks().get(bankCase5).getClients()
    .get(clientCase5).
150                             getAccounts().get(accountCase5).takeMoney(
    amountCase5);
151                 }
152                 case "6" -> {
153                     IAccount from;
154                     IAccount to;
155                     System.out.println("Choose in what bank client
    registered:");
156                     for (Bank bank : centralBank.getBanks()) {
157                         System.out.println(bank.getName() + " ");
158                     }
159                     System.out.println();
160                     Scanner inCase6 = new Scanner(System.in);
161                     int bankCase6 = inCase6.nextInt();
162                     System.out.println("Choose account's owner");
163                     for (Client client : centralBank.getBanks().get(
    bankCase6).getClients()) {
164                         System.out.println(client.getId() + " ");
165                     }
166                     System.out.println();
167                     int clientCase6 = inCase6.nextInt();
168                     System.out.println("Choose account:");
169                     for (IAccount account :
```

```
170                         centralBank.getBanks().get(bankCase6).
    getClients().get(clientCase6).getAccounts()) {
171                     System.out.println(account.getId() + " ");
172                 }
173                 System.out.println();
174                 int accountCase6 = inCase6.nextInt();
175                 from = centralBank.getBanks().get(bankCase6).
    getClients().get(clientCase6).
176                         getAccounts().get(accountCase6);
177                 System.out.println("Enter amount:");
178                 double amountCase6 = inCase6.nextDouble();
179                 centralBank.getBanks().get(bankCase6).getClients()
    .get(clientCase6).
180                         getAccounts().get(accountCase6).takeMoney(
    amountCase6);
181                 System.out.println("Choose in what bank client
    registered:");
182                 for (Bank bank : centralBank.getBanks()) {
183                     System.out.println(bank.getName() + " ");
184                 }
185                 System.out.println();
186                 bankCase6 = inCase6.nextInt();
187                 System.out.println("Choose account's owner");
188                 for (Client client : centralBank.getBanks().get(
    bankCase6).getClients()) {
189                     System.out.println(client.getId() + " ");
190                 }
191                 System.out.println();
192                 clientCase6 = inCase6.nextInt();
193                 System.out.println("Choose account:");
194                 for (IAccount account :
195                         centralBank.getBanks().get(bankCase6).
    getClients().get(clientCase6).getAccounts()) {
196                     System.out.println(account.getId() + " ");
197                 }
198                 System.out.println();
199                 accountCase6 = inCase6.nextInt();
200                 to = centralBank.getBanks().get(bankCase6).
    getClients().get(clientCase6).
201                         getAccounts().get(accountCase6);
202                 centralBank.getBanks().get(bankCase6).getClients()
    .get(clientCase6).
203                         getAccounts().get(accountCase6).addMoney(
    amountCase6);
204                 centralBank.makeTransaction(from, amountCase6, to)
    ;
205             }
206             case "0" -> System.exit(0);
207             default -> {
208                 var e = new IllegalArgumentException();
```

```
209                     throw new BankException("Invalid choice!", e);
210                 }
211             }
212         }
213     }
214 }
```

Листинг 1.2: Main.java

```
1  import bank.CentralBank;
2  import tools.BankException;
3
4  public class Main {
5      public static void main(String[] args) throws BankException {
6          CentralBank cb = new CentralBank();
7          Console ui = new Console(cb);
8          ui.work();
9      }
10 }
```

Листинг 1.3: AccountType.java

```
1 package account;
2
3 public enum AccountType {
4     Debit,
5     Deposit,
6     Credit,
7 }
```

Листинг 1.4: Credit.java

```
1  package account;
2
3  import tools.BankException;
4
5  import java.util.UUID;
6
7  public class Credit implements IAccount {
8      private double fee;
9      private double limit;
10
11     private double unverifiedLimit;
12     private boolean verified;
13
14     private double accumulatedFee;
15     private double balance;
16
17     private UUID id;
18
19     public Credit(double fee, double limit, boolean verified, double
       unverifiedLimit) throws BankException {
20         if (fee <= 0) throw new BankException("Fee for credit account
       must be positive!");
21         if (limit <= 0) throw new BankException("Limit for credit
       account must be positive!");
22         if (unverifiedLimit <= 0) throw new BankException("Limit for
       unverified account must be positive!");
23
24         this.fee = fee;
25         this.limit = limit;
26         this.unverifiedLimit = unverifiedLimit;
27         this.verified = verified;
28
29         id = UUID.randomUUID();
30
31         accumulatedFee = 0;
32         balance = 0;
33     }
34
35     @Override
36     public UUID getId() {
37         return id;
38     }
39
40     @Override
41     public double getBalance() {
42         return balance;
43     }
44
45     public void takeMoney(double amount) throws BankException {
```

```
46          if (amount <= 0) throw new BankException("Amount must be
    positive!");
47          if (amount > balance + limit) throw new BankException("Amount
    is too big!");
48          if (amount > unverifiedLimit && !verified)
49              throw new BankException("Amount is bigger than limit for
    unverified account!");
50
51          balance -= amount;
52      }
53
54      public void addMoney(double amount) throws BankException {
55          if (amount <= 0) throw new BankException("Amount must be
    positive!");
56          balance += amount;
57      }
58
59      public void calculateDailyPayment() {
60          if (balance < 0) {
61              accumulatedFee += fee;
62          }
63      }
64
65      public void getReward() throws BankException {
66          if (balance + limit < accumulatedFee) throw new BankException(
    "Balance too low to take commission!");
67          balance -= accumulatedFee;
68          accumulatedFee = 0;
69      }
70
71      @Override
72      public boolean equals(Object obj) {
73          if (obj == this) return true;
74          if (obj == null || obj.getClass() != this.getClass()) return
    false;
75
76          Credit other = (Credit) obj;
77
78          return other.getId() == this.getId();
79      }
80
81      @Override
82      public int hashCode() {
83          return id.hashCode();
84      }
85  }
```

Листинг 1.5: Debit.java

```
1  package account;
2
3  import tools.BankException;
4
5  import java.util.UUID;
6
7  public class Debit implements IAccount {
8      private double interest;
9
10     private double unverifiedLimit;
11     private boolean verified;
12
13     private double accumulatedAmount;
14     private double balance;
15
16     private UUID id;
17
18     public Debit(double interest, boolean verified, double
       unverifiedLimit) throws BankException {
19         if (interest <= 0) throw new BankException("Interest for debit
        account must be positive!");
20         if (unverifiedLimit <= 0) throw new BankException("Limit for
       unverified account must be positive!");
21
22         this.interest = interest;
23         this.verified = verified;
24         this.unverifiedLimit = unverifiedLimit;
25
26         id = UUID.randomUUID();
27
28         accumulatedAmount = 0;
29         balance = 0;
30     }
31
32     @Override
33     public UUID getId() {
34         return id;
35     }
36
37     @Override
38     public double getBalance() {
39         return balance;
40     }
41
42     public void takeMoney(double amount) throws BankException {
43         if (amount <= 0) throw new BankException("Amount must be
       positive!");
44         if (amount > balance) throw new BankException("Amount is too
       big!");
```

```
45          if (amount > unverifiedLimit && !verified)
46              throw new BankException("Amount is bigger than limit for
    unverified account!");
47
48          balance -= amount;
49      }
50
51      public void addMoney(double amount) throws BankException {
52          if (amount <= 0) throw new BankException("Amount must be
    positive!");
53          balance += amount;
54      }
55
56      @Override
57      public void calculateDailyPayment() {
58          accumulatedAmount += balance * interest / 365;
59      }
60
61      @Override
62      public void getReward() {
63          balance += accumulatedAmount;
64          accumulatedAmount = 0;
65      }
66
67      @Override
68      public boolean equals(Object obj) {
69          if (obj == this) return true;
70          if (obj == null || obj.getClass() != this.getClass()) return
    false;
71
72          Debit other = (Debit) obj;
73
74          return other.getId() == this.getId();
75      }
76
77      @Override
78      public int hashCode() {
79          return id.hashCode();
80      }
81  }
```

Листинг 1.6: Deposit.java

```
1  package account;
2
3  import tools.BankException;
4
5  import java.time.LocalDate;
6  import java.util.Map;
7  import java.util.UUID;
8
9  public class Deposit implements IAccount {
10     private double interest;
11
12     private double unverifiedLimit;
13     private boolean verified;
14
15     private LocalDate validUntil;
16
17     private double accumulatedAmount;
18     private double balance;
19
20     private UUID id;
21
22     public Deposit(double interest, Map<Double, Double>
    interestConditions,
23                    DepositDTO depositData, boolean verified, double
    unverifiedLimit) throws BankException {
24         if (interest <= 0) throw new BankException("Interest for debit
     account must be positive!");
25         if (unverifiedLimit <= 0) throw new BankException("Limit for
    unverified account must be positive!");
26
27         for (Double percent : interestConditions.keySet()) {
28             if (percent <= 0) throw new BankException("Percent can not
     be null!");
29         }
30
31         for (Double amount : interestConditions.values()) {
32             if (amount <= 0) throw new BankException("Amount can not
    be null!");
33         }
34
35         if (depositData == null) {
36             var e = new IllegalArgumentException();
37             throw new BankException("Deposit data can not be null!", e
    );
38         }
39
40         balance = depositData.getBalance();
41         validUntil = LocalDate.parse(depositData.getValidUntil());
42
```

```
43          if (balance <= 0) throw new BankException("Balance must be
    positive!");
44          if (validUntil.isBefore(LocalDate.now())) throw new
    BankException("Invalid date!");
45
46          this.interest = interest;
47          for (Map.Entry<Double, Double> condition : interestConditions.
    entrySet()) {
48              if (balance <= condition.getKey()) this.interest =
    condition.getValue();
49          }
50
51          this.verified = verified;
52          this.unverifiedLimit = unverifiedLimit;
53
54          id = UUID.randomUUID();
55          accumulatedAmount = 0;
56      }
57
58      @Override
59      public UUID getId() {
60          return id;
61      }
62
63      @Override
64      public double getBalance() {
65          return balance;
66      }
67
68      @Override
69      public void takeMoney(double amount) throws BankException {
70          if (amount <= 0) throw new BankException("Amount must be
    positive!");
71          if (amount > balance) throw new BankException("Amount is too
    big!");
72          if (amount > unverifiedLimit && !verified)
73              throw new BankException("Amount is bigger than limit for
    unverified account!");
74
75          if (LocalDate.now().isBefore(validUntil)) throw new
    BankException("It is impossible to take money now!");
76
77          balance -= amount;
78      }
79
80      @Override
81      public void addMoney(double amount) throws BankException {
82          if (amount <= 0) throw new BankException("Amount must be
    positive!");
83          balance += amount;
```

```
84      }
85
86      @Override
87      public void calculateDailyPayment() {
88          accumulatedAmount += balance * interest / 365;
89      }
90
91      @Override
92      public void getReward() {
93          balance += accumulatedAmount;
94          accumulatedAmount = 0;
95      }
96
97      @Override
98      public boolean equals(Object obj) {
99          if (obj == this) return true;
100         if (obj == null || obj.getClass() != this.getClass()) return
        false;
101
102         Deposit other = (Deposit) obj;
103
104         return other.getId() == this.getId();
105     }
106
107     @Override
108     public int hashCode() {
109         return id.hashCode();
110     }
111 }
```

Листинг 1.7: DepositConsole.java

```
1  package account;
2
3  import java.util.Scanner;
4
5  public class DepositConsole {
6      public DepositDTO collectDepositConditions() {
7          Scanner in = new Scanner(System.in);
8
9          System.out.println("Enter amount you want to deposit:");
10         double balance = in.nextDouble();
11
12         System.out.println("Enter date you want deposit will be valid
       until (dd-mm-yyyy):");
13         String date = in.nextLine();
14
15         return new DepositDTO(balance, date);
16     }
17 }
```

Листинг 1.8: DepositDTO.java

```
package account;

public class DepositDTO {
    private double balance;
    private String validUntil;

    public DepositDTO(double balance, String validUntil){
        this.balance = balance;
        this.validUntil = validUntil;
    }

    public double getBalance() {
        return balance;
    }

    public String getValidUntil() {
        return validUntil;
    }
}
```

Листинг 1.9: IAccount.java

```
1  package account;
2
3  import tools.BankException;
4
5  import java.util.UUID;
6
7  public interface IAccount {
8
9      double getBalance();
10     UUID getId();
11
12     void takeMoney(double amount) throws BankException;
13     void addMoney(double amount) throws BankException;
14
15     void calculateDailyPayment();
16     void getReward() throws BankException;
17
18 }
```

Листинг 1.10: Bank.java

```
1  package bank;
2
3  import account.*;
4  import client.Client;
5  import client.ClientDTO;
6  import tools.BankException;
7  import tools.EventManager;
8  import tools.IEventListener;
9
10 import java.util.ArrayList;
11 import java.util.List;
12 import java.util.Map;
13
14 public class Bank implements IEventListener {
15
16     private CentralBank centralBank;
17     private List<Client> clients;
18
19     private String name;
20     private double debitInterest;
21     private double creditFee;
22     private double creditLimit;
23     private double unverifiedLimit;
24     private double depositDefaultInterest;
25     private Map<Double, Double> depositInterestConditions;
26
27     public EventManager events;
28
29     public Bank(CentralBank centralBank, BankDTO bankData) throws
       BankException {
30         if (centralBank == null) {
31             var e = new IllegalArgumentException();
32             throw new BankException("Central bank can not be null!", e
       );
33         }
34
35         if (bankData == null) {
36             var e = new IllegalArgumentException();
37             throw new BankException("Bank data can not be null!", e);
38         }
39
40         if (bankData.getName() == null) {
41             var e =  new IllegalArgumentException();
42             throw new BankException("Bank data can not be null!", e);
43         }
44         if (bankData.getCreditFee() <= 0) throw new BankException("
       Credit fee can not be negative!");
45         if (bankData.getCreditLimit() <= 0) throw new BankException("
       Credit limit can not be negative!");
```

```
46          if (bankData.getDebitInterest() <= 0) throw new BankException(
    "Debit interest can not be negative!");
47          if (bankData.getDepositDefaultInterest() <= 0) throw new
    BankException("Deposit default interest can not be negative!");
48          if (bankData.getUnverifiedLimit() <= 0) throw new
    BankException("Unverified limit can not be negative!");
49
50          this.centralBank = centralBank;
51
52          name = bankData.getName();
53
54          debitInterest = bankData.getDebitInterest();
55          creditFee = bankData.getCreditFee();
56          creditLimit = bankData.getCreditLimit();
57          unverifiedLimit = bankData.getUnverifiedLimit();
58          depositDefaultInterest = bankData.getDepositDefaultInterest();
59          depositInterestConditions = bankData.
    getDepositInterestConditions();
60
61          clients = new ArrayList<>();
62
63          events = new EventManager("unverified limit", "debit interest"
    , "credit fee", "credit limit");
64      }
65
66      public String getName() {
67          return name;
68      }
69
70      public double getDebitInterest() {
71          return debitInterest;
72      }
73
74      public double getCreditFee() {
75          return creditFee;
76      }
77
78      public double getCreditLimit() {
79          return creditLimit;
80      }
81
82      public double getUnverifiedLimit() {
83          return unverifiedLimit;
84      }
85
86      public double getDepositDefaultInterest() {
87          return depositDefaultInterest;
88      }
89
90      public List<Client> getClients() {
```

```
91          return clients;
92      }
93
94      public Client registerClient(ClientDTO clientData) throws
    BankException {
95          if (clientData == null) {
96              var e = new IllegalArgumentException();
97              throw new BankException("Client data can not be null!", e)
    ;
98          }
99
100         for (Client client: clients) {
101             if (client.getId() == clientData.getId()) {
102                 throw new BankException("Such client already exist!");
103             }
104         }
105
106         Client client = new Client(clientData, this);
107         clients.add(client);
108
109         return client;
110     }
111
112     public Client fillMissingData(Client client, ClientDTO clientData)
     throws BankException {
113         if (client == null) {
114             var e = new IllegalArgumentException();
115             throw new BankException("Client can not be null!", e);
116         }
117
118         if (clientData == null) {
119             var e = new IllegalArgumentException();
120             throw new BankException("Client data can not be null!", e)
    ;
121         }
122
123         if (!clients.contains(client)) throw new BankException("
    Unknown client!");
124
125         client.addMissingData(clientData);
126
127         return client;
128     }
129
130     public IAccount registerAccount(Client client, AccountType
    accountType, DepositDTO depositData) throws BankException {
131         if (client == null) {
132             var e = new IllegalArgumentException();
133             throw new BankException("Client can not be null!", e);
134         }
```

```
135
136         IAccount account;
137         switch (accountType) {
138             case Credit -> account = new Credit(creditFee, creditLimit
    , client.getVerified(), unverifiedLimit);
139             case Debit -> account = new Debit(debitInterest, client.
    getVerified(), unverifiedLimit);
140             case Deposit -> account = new Deposit(
    depositDefaultInterest, depositInterestConditions, depositData,
    client.getVerified(), unverifiedLimit);
141             default -> {
142                 var e = new IllegalArgumentException();
143                 throw new BankException("Invalid account type!", e);
144             }
145         }
146
147         client.addAccount(account);
148
149         return account;
150     }
151
152     public void calculateDailyPayment() {
153         for (Client client: clients) {
154             for (IAccount account: client.getAccounts()) {
155                 account.calculateDailyPayment();
156             }
157         }
158     }
159
160     public void payReward() throws BankException {
161         for (Client client: clients) {
162             for (IAccount account: client.getAccounts()) {
163                 account.getReward();
164             }
165         }
166     }
167
168     public void changeUnverifiedLimit(double newLimit) throws
    BankException {
169         if (newLimit <= 0) throw new BankException("Limit must be
    positive!");
170         unverifiedLimit = newLimit;
171
172         events.notify("unverified limit");
173     }
174
175     public void changeDebitInterest(double newInterest) throws
    BankException {
176         if (newInterest <= 0) throw new BankException("Interest must
    be positive!");
```

```
177         debitInterest = newInterest;
178
179         events.notify("debit interest");
180     }
181
182     public void changeCreditFee(double newFee) throws BankException {
183         if (newFee <= 0) throw new BankException("Fee must be positive
    !");
184         creditFee = newFee;
185
186         events.notify("credit fee");
187     }
188
189     public void changeCreditLimit(double newLimit) throws
    BankException {
190         if (newLimit <= 0) throw new BankException("Limit must be
    positive!");
191         creditFee = newLimit;
192
193         events.notify("credit limit");
194     }
195
196     @Override
197     public void update(String eventType) throws BankException {
198         switch (eventType) {
199             case "daily payment" -> calculateDailyPayment();
200             case "monthly payment" -> payReward();
201             default -> {
202                 var e = new IllegalArgumentException();
203                 throw new BankException("Invalid event!", e);
204             }
205         }
206     }
207
208     @Override
209     public boolean equals(Object obj) {
210         if (obj == this) return true;
211         if (obj == null || obj.getClass() != this.getClass()) return
    false;
212
213         Bank other = (Bank) obj;
214
215         return other.getName().equals(this.getName());
216     }
217
218     @Override
219     public int hashCode() {
220         return name.hashCode();
221     }
222 }
```

Листинг 1.11: BankConsole.java

```
1  package bank;
2
3  import java.util.HashMap;
4  import java.util.Scanner;
5
6  public class BankConsole {
7      public BankDTO collectBankData() {
8          Scanner in = new Scanner(System.in);
9
10         System.out.println("Enter bank name:");
11         String name = in.nextLine();
12
13         System.out.println("Enter debit interest:");
14         double debitInterest = in.nextDouble();
15
16         System.out.println("Enter credit fee:");
17         double creditFee = in.nextDouble();
18
19         System.out.println("Enter credit limit:");
20         double creditLimit = in.nextDouble();
21
22         System.out.println("Enter limit for unverified clients:");
23         double unverifiedLimit = in.nextDouble();
24
25         System.out.println("Enter deposit default interest:");
26         double depositDefaultInterest = in.nextDouble();
27
28         System.out.println("Enter how many conditions will be");
29         int n = in.nextInt();
30
31         HashMap<Double, Double> conditions = new HashMap<>(n);
32
33         for (int i = 0; i < n; i++) {
34             System.out.println("Enter amount border:");
35             double amountBorder = in.nextDouble();
36
37             System.out.println("Enter interest for this border");
38             double interestBorder = in.nextDouble();
39
40             conditions.put(amountBorder, interestBorder);
41         }
42
43         return new BankDTO(name, debitInterest, creditFee, creditLimit
       ,
44                 unverifiedLimit, depositDefaultInterest, conditions);
45     }
46 }
```

Листинг 1.12: BankDTO.java

```
1  package bank;
2
3  import java.util.Map;
4
5  public class BankDTO {
6
7      private String name;
8      private double debitInterest;
9      private double creditFee;
10     private double creditLimit;
11     private double unverifiedLimit;
12     private double depositDefaultInterest;
13     private Map<Double, Double> depositInterestConditions;
14
15     public BankDTO(String name, double debitInterest, double creditFee
       , double creditLimit,
16                    double unverifiedLimit, double
       depositDefaultInterest, Map<Double, Double>
       depositInterestConditions) {
17         this.name = name;
18         this.debitInterest = debitInterest;
19         this.creditFee = creditFee;
20         this.creditLimit = creditLimit;
21         this.unverifiedLimit = unverifiedLimit;
22         this.depositDefaultInterest = depositDefaultInterest;
23         this.depositInterestConditions = depositInterestConditions;
24     }
25
26     public double getCreditFee() {
27         return creditFee;
28     }
29
30     public double getCreditLimit() {
31         return creditLimit;
32     }
33
34     public double getDebitInterest() {
35         return debitInterest;
36     }
37
38     public String getName() {
39         return name;
40     }
41
42     public double getDepositDefaultInterest() {
43         return depositDefaultInterest;
44     }
45
46     public double getUnverifiedLimit() {
```

```
47          return unverifiedLimit;
48      }
49
50      public Map<Double, Double> getDepositInterestConditions() {
51          return depositInterestConditions;
52      }
53  }
```

Листинг 1.13: CentralBank.java

```
1  package bank;
2
3  import account.IAccount;
4  import tools.BankException;
5  import tools.EventManager;
6
7  import java.time.Duration;
8  import java.time.LocalDate;
9  import java.util.ArrayList;
10 import java.util.List;
11 import java.util.Objects;
12
13 public class CentralBank {
14
15     private final List<Bank> banks;
16     private final List<Transaction> transactions;
17
18     public EventManager events;
19
20     public CentralBank() {
21         banks = new ArrayList<>();
22         transactions = new ArrayList<>();
23
24         events = new EventManager("daily payment", "monthly payment");
25     }
26
27     public List<Bank> getBanks() {
28         return banks;
29     }
30
31     public List<Transaction> getTransactions() {
32         return transactions;
33     }
34
35     public Bank addBank(BankDTO bankData) throws BankException {
36         if (bankData == null) {
37             var e = new IllegalArgumentException();
38             throw new BankException("Bank can not be null!", e);
39         }
40
41         for (Bank bank: banks) {
42             if (Objects.equals(bank.getName(), bankData.getName())) {
43                 throw new BankException("Bank with such name already
    exist!");
44             }
45         }
46
47         Bank bank = new Bank(this, bankData);
48         banks.add(bank);
```

```
49
50          return  bank;
51      }
52
53      public Transaction makeTransaction(IAccount from, double amount,
    IAccount to) throws BankException {
54          if (from == null) {
55              var e = new IllegalArgumentException();
56              throw new BankException("Account can not be null!", e);
57          }
58
59          if (to == null) {
60              var e = new IllegalArgumentException();
61              throw new BankException("Account can not be null!", e);
62          }
63
64          if (amount <= 0) throw new BankException("Amount can not be
    negative!");
65
66          from.takeMoney(amount);
67          to.addMoney(amount);
68
69          var transaction = new Transaction(from, amount, to);
70          transactions.add(transaction);
71
72          return transaction;
73      }
74
75      public void cancelTransaction(Transaction transaction) throws
    BankException {
76          if (transaction == null) {
77              var e = new IllegalArgumentException();
78              throw new BankException("Transaction can not be null!", e)
    ;
79          }
80
81          transaction.getFrom().addMoney(transaction.getAmount());
82          transaction.getTo().takeMoney(transaction.getAmount());
83
84          transactions.remove(transaction);
85      }
86
87      public void calculateIncome(LocalDate from, LocalDate to) throws
    BankException {
88          long daysBetween = Duration.between(from, to).toDays();
89          for (long i = 0; i < daysBetween; i++) {
90              events.notify("daily payment");
91              if (i % 30 == 0 && i > 0) events.notify("monthly payment")
    ;
92          }
```

```
93          }
94
95
96  }
```

Листинг 1.14: Transaction.java

```
package bank;

import account.IAccount;
import tools.BankException;

public class Transaction {
    private final double amount;
    private final IAccount from;
    private final IAccount to;

    public Transaction(IAccount from, double amount, IAccount to)
   throws BankException {
        if (from == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Account can not be null!", e);
        }

        if (to == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Account can not be null!", e);
        }

        if (amount <= 0) throw new BankException("Amount can not be
   negative!");

        this.amount = amount;
        this.from = from;
        this.to = to;
    }

    public double getAmount() {
        return amount;
    }

    public IAccount getFrom() {
        return from;
    }

    public IAccount getTo() {
        return to;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == this) return true;
        if (obj == null || obj.getClass() != this.getClass()) return
   false;

        Transaction other = (Transaction) obj;
```

```
47
48          return from.equals(other.from) && to.equals(other.to) &&
        amount == other.amount;
49      }
50
51      @Override
52      public int hashCode() {
53          return from.hashCode() + to.hashCode() - (int) amount;
54      }
55  }
```

Листинг 1.15: Client.java

```
1  package client;
2
3  import account.IAccount;
4  import bank.Bank;
5  import tools.BankException;
6  import tools.IEventListener;
7
8  import java.util.ArrayList;
9  import java.util.List;
10 import java.util.UUID;
11
12 public class Client implements IEventListener {
13
14     private final String name;
15     private final String surname;
16
17     private String address;
18     private String passport;
19
20     private final UUID id;
21
22     private boolean verified;
23
24     private final List<IAccount> accounts;
25
26     private final Bank bank;
27
28     public Client(ClientDTO clientData, Bank bank) throws
       BankException {
29
30         if (clientData == null){
31             var e = new IllegalArgumentException();
32             throw new BankException("Client data can not be null!", e)
       ;
33         }
34
35         if (bank == null) {
36             var e = new IllegalArgumentException();
37             throw new BankException("Bank can not be null!", e);
38         }
39
40         this.bank = bank;
41
42         if (clientData.getName() == null) throw new
       IllegalArgumentException();
43         if (clientData.getSurname() == null) throw new
       IllegalArgumentException();
44
45         name = clientData.getName();
```

```
46          surname = clientData.getSurname();
47
48          if (clientData.getAddress() == null) throw new
        IllegalArgumentException();
49          if (clientData.getPassport() == null) throw new
        IllegalArgumentException();
50
51          verified = !clientData.getAddress().equals("LATER") && !
        clientData.getPassport().equals("LATER");
52
53          address = clientData.getAddress();
54          passport = clientData.getPassport();
55
56          id = clientData.getId();
57          accounts = new ArrayList<>();
58      }
59
60      public String getName() {
61          return name;
62      }
63
64      public String getSurname() {
65          return surname;
66      }
67
68      public String getAddress() {
69          return address;
70      }
71
72      public String getPassport() {
73          return passport;
74      }
75
76      public UUID getId() {
77          return id;
78      }
79
80      public boolean getVerified() {
81          return verified;
82      }
83
84      public List<IAccount> getAccounts() {
85          return accounts;
86      }
87
88      public void addMissingData(ClientDTO clientData) throws
        BankException {
89          if (clientData == null){
90              var e = new IllegalArgumentException();
91              throw new BankException("Client data can not be null!", e)
```

```
    ;
92          }
93
94          if (clientData.getAddress() == null) throw new
    IllegalArgumentException();
95          if (clientData.getPassport() == null) throw new
    IllegalArgumentException();
96
97          if (clientData.getAddress().equals("LATER")) throw new
    BankException("Address must be valid!");
98          if (clientData.getPassport().equals("LATER")) throw new
    BankException("Passport must be valid!");
99
100         address = clientData.getAddress();
101         passport = clientData.getPassport();
102
103         verified = true;
104     }
105
106     public void addAccount(IAccount account) throws BankException {
107         if (account == null) {
108             var e = new IllegalArgumentException();
109             throw new BankException("Account can not be null!", e);
110         }
111
112         if (accounts.contains(account)) throw new BankException("This
    client already has this account!");
113         accounts.add(account);
114     }
115
116     public void displayEvent(String event) {
117     }
118
119     @Override
120     public void update(String eventType) throws BankException {
121         displayEvent(eventType);
122     }
123
124     @Override
125     public boolean equals(Object obj) {
126         if (obj == this) return true;
127         if (obj == null || obj.getClass() != this.getClass()) return
    false;
128
129         Client other = (Client) obj;
130
131         return other.getId() == this.getId();
132     }
133
134     @Override
```

```
135     public int hashCode() {
136         return id.hashCode();
137     }
138 }
```

Листинг 1.16: ClientConsole.java

```
1  package client;
2
3  import tools.BankException;
4
5  import java.util.Scanner;
6
7  public class ClientConsole {
8      public ClientDTO collectPersonalData() {
9          Scanner in = new Scanner(System.in);
10
11         System.out.println("Enter your name:");
12         String name = in.nextLine();
13
14         System.out.println("Enter your surname:");
15         String surname = in.nextLine();
16
17         System.out.println("Enter your address:");
18         String address = in.nextLine();
19
20         System.out.println("Enter your passport:");
21         String passport = in.nextLine();
22
23         return new ClientDTO(name, surname, address, passport);
24     }
25
26     public ClientDTO addMissingData(ClientDTO clientData) throws
       BankException {
27         if (clientData == null) {
28             var e = new IllegalArgumentException();
29             throw new BankException("Client data can not be null!", e)
       ;
30         }
31
32         String address = "LATER";
33         String passport = "LATER";
34         Scanner in = new Scanner(System.in);
35
36         if (clientData.getAddress().equals("LATER")) {
37             System.out.println("Enter your address^");
38             address = in.nextLine();
39         }
40
41         if (clientData.getPassport().equals("LATER")) {
42             System.out.println("Enter your passport");
43             passport = in.nextLine();
44         }
45
46         return new ClientDTO(clientData.getName(), clientData.
       getSurname(), address, passport);
```

```
47         }
48 }
```

Листинг 1.17: ClientDTO.java

```
1  package client;
2
3  import java.util.UUID;
4
5  public class ClientDTO {
6
7      private final String name;
8      private final String surname;
9      private final String address;
10     private final String passport;
11     private final UUID id;
12
13     public ClientDTO(String name, String surname, String address,
       String passport) {
14         this.name = name;
15         this.surname = surname;
16         this.address = address;
17         this.passport = passport;
18
19         id = UUID.randomUUID();
20     }
21
22     public String getName() {
23         return name;
24     }
25
26     public String getSurname() {
27         return surname;
28     }
29
30     public String getAddress() {
31         return address;
32     }
33
34     public String getPassport() {
35         return passport;
36     }
37
38     public UUID getId() {
39         return id;
40     }
41 }
```

Листинг 1.18: BankException.java

```
1  package tools;
2
3  public class BankException extends Exception {
4
5      public BankException() {
6          super();
7      }
8
9      public BankException(String message) {
10         super(message);
11     }
12
13     public BankException(String message, Throwable cause) {
14         super(message, cause);
15     }
16 }
```

Листинг 1.19: EventManager.java

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

Листинг 1.20: IEventListener.java

```
1 package tools;
2
3 public interface IEventListener {
4     void update(String eventType) throws BankException;
5 }
```