Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Лабороторная работа №3

Выполнил студент:

Жарков Григорий Алексеевич
группа: M32071

Проверил:

Чикишев Константин Максимович

Санкт-Петербург,
2022 г.

1.1. Текст задания

3 лабораторная
К созданному в прошлой лабораторной сервису добавляется Spring.
Сервис должен предоставлять http интерфейс (REST API) для получения информации о конкретных котиках и владельцах и для получения фильтрованной информации (например, получить всех рыжих котиков)
Внимание: недопустимо отдавать через HTTP интерфейс сущности JPA. Рекомендуется создать отдельные оберточные классы.
Сервисы и dao должны превратиться в Spring Bean'ы с использованием Dependency Injection (Autowired). Dao при этом наследуют JpaRepository и имеет шаблонные Spring Data JPA методы: https://www.baeldung.com/spring-data-repositoriesrepos
При сдаче лабораторной нужно будет показать работоспособность endpoint'ов через http запросы (рекомендуется Postman).
В рамках лабораторной к проекту должен быть добавлен CI/CD, запускающий тесты проекта kotiki-java.

1.2. Решение

Листинг 1.1: Console.java

```
import account.AccountType;
import account.DepositConsole;
import account.DepositDTO;
import account.IAccount;
import bank.Bank;
import bank.BankConsole;
import bank.BankDTO;
import bank.CentralBank;
import client.Client;
import client.ClientConsole;
import client.ClientDTO;
import tools.BankException;

import java.util.Scanner;

public class Console {
    private CentralBank centralBank;

    public Console(CentralBank centralBank) throws BankException {
        if (centralBank == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Central bank can not be null!", e);
        }

        this.centralBank = centralBank;
    }

    public void work() throws BankException {
        while (true) {
            System.out.println("Enter what you want to do: " +
                    "\n1) Add new bank " +
                    "\n2) Add new client " +
                    "\n3) Add new account " +
                    "\n4) Add money to account " +
                    "\n5) Take money from account " +
                    "\n6) Make transaction " +
                    "\n0) Exit");

            Scanner choice = new Scanner(System.in);
            String answer = choice.nextLine();

            switch (answer) {
                case "1" -> {
                    BankDTO bankData = new BankConsole().collectBankData();
                    centralBank.addBank(bankData);
                }
                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

```
1  package bank;
2
3  import account.IAccount;
4  import tools.BankException;
5
6  public class Transaction {
7      private final double amount;
8      private final IAccount from;
9      private final IAccount to;
10
11     public Transaction(IAccount from, double amount, IAccount to)
       throws BankException {
12         if (from == null) {
13             var e = new IllegalArgumentException();
14             throw new BankException("Account can not be null!", e);
15         }
16
17         if (to == null) {
18             var e = new IllegalArgumentException();
19             throw new BankException("Account can not be null!", e);
20         }
21
22         if (amount <= 0) throw new BankException("Amount can not be
       negative!");
23
24         this.amount = amount;
25         this.from = from;
26         this.to = to;
27     }
28
29     public double getAmount() {
30         return amount;
31     }
32
33     public IAccount getFrom() {
34         return from;
35     }
36
37     public IAccount getTo() {
38         return to;
39     }
40
41     @Override
42     public boolean equals(Object obj) {
43         if (obj == this) return true;
44         if (obj == null || obj.getClass() != this.getClass()) return
       false;
45
46         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

```
package client;

import java.util.UUID;

public class ClientDTO {

    private final String name;
    private final String surname;
    private final String address;
    private final String passport;
    private final UUID id;

    public ClientDTO(String name, String surname, String address,
   String passport) {
        this.name = name;
        this.surname = surname;
        this.address = address;
        this.passport = passport;

        id = UUID.randomUUID();
    }

    public String getName() {
        return name;
    }

    public String getSurname() {
        return surname;
    }

    public String getAddress() {
        return address;
    }

    public String getPassport() {
        return passport;
    }

    public UUID getId() {
        return id;
    }
}
```

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 }
```

Листинг 1.21: Application.java

```
1  package com.griga;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  @SpringBootApplication
7  public class Application {
8      public static void main(String[] args) {
9          SpringApplication.run(Application.class, args);
10     }
11 }
```

Листинг 1.22: CatController.java

```
1  package com.griga.controller.controllers;
2
3  import com.griga.controller.tools.ControllerException;
4  import com.griga.dao.colors.Color;
5  import com.griga.dto.CatDTO;
6  import com.griga.service.tools.ServiceException;
7  import org.springframework.beans.factory.annotation.Autowired;
8  import org.springframework.http.HttpStatus;
9  import org.springframework.web.bind.annotation.*;
10 import com.griga.service.SpringService;
11 import org.springframework.web.server.ResponseStatusException;
12
13 import java.util.List;
14
15 @RestController
16 @RequestMapping("/cat")
17 public class CatController {
18     @Autowired
19     private SpringService service;
20
21     @GetMapping("/get-all")
22     public List<CatDTO> getAllCats() {
23         return service.getAllCats();
24     }
25
26     @PostMapping("/add")
27     public boolean addCat(@RequestBody CatDTO catDTO) throws
    ControllerException {
28         try {
29             return service.addCat(catDTO);
30         } catch (ServiceException e) {
31             throw new ControllerException("Problem with adding cat", e
    );
32         }
33     }
34
35     @DeleteMapping("/delete/{id}")
36     public boolean removeCat(@PathVariable("id") String catId) throws
    ControllerException {
37         try {
38             return service.removeCat(Long.valueOf(catId));
39         } catch (ServiceException e) {
40             throw new ControllerException("Problem with deleting cat",
     e);
41
42         }
43     }
44
45     @PostMapping("/add-to-owner/{id}")
```

```
46      public boolean addToOwner(@RequestBody CatDTO catDTO,
    @PathVariable("id") String ownerId) throws ControllerException {
47          try {
48              return service.addCatToOwner(catDTO, Long.valueOf(ownerId)
    );
49          } catch (ServiceException e) {
50              throw new ControllerException("Problem with adding cat to
    owner", e);
51          }
52      }
53
54      @DeleteMapping("/remove-from-owner/{id}")
55      public boolean removeFromOwner(@RequestBody CatDTO catDTO,
    @PathVariable("id") String ownerId) throws ControllerException {
56          try {
57              return service.removeCatFromOwner(catDTO, Long.valueOf(
    ownerId));
58          } catch (ServiceException e) {
59              throw new ControllerException("Problem with removing cat
    from owner", e);
60          }
61      }
62
63      @GetMapping("get-by-color/{color}")
64      public List<CatDTO> getByColor(@PathVariable("color") String color
    ) {
65          try {
66              return service.getCatsByColor(Color.valueOf(color));
67          } catch (IllegalArgumentException e) {
68              throw new ResponseStatusException(HttpStatus.BAD_REQUEST);
69          }
70      }
71
72      @PostMapping("/make-friends/{id}")
73      public boolean makeFriendship(@RequestBody CatDTO firstCatDTO,
    @PathVariable("id") String secondCatId) throws ControllerException
    {
74          try {
75              return service.makeFriendship(firstCatDTO, Long.valueOf(
    secondCatId));
76          } catch (ServiceException e) {
77              throw new ControllerException("Problem with making
    friendship", e);
78          }
79      }
80
81      @DeleteMapping("/break-friends/{id}")
82      public boolean breakFriendship(@RequestBody CatDTO firstCatDTO,
    @PathVariable("id") String secondCatId) throws ControllerException
    {
```

```
83          try {
84              return service.breakFriendship(firstCatDTO, Long.valueOf(
    secondCatId));
85          } catch (ServiceException e) {
86              throw new ControllerException("Problem with breaking
    friendship", e);
87          }
88      }
89
90  }
```

Листинг 1.23: OwnerController.java

```
package com.griga.controller.controllers;

import com.griga.controller.tools.ControllerException;
import com.griga.dto.OwnerDTO;
import com.griga.service.tools.ServiceException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import com.griga.service.SpringService;

import java.util.List;

@RestController
@RequestMapping("/owner")
public class OwnerController {
    @Autowired
    private SpringService service;

    @GetMapping("/get-all")
    public List<OwnerDTO> getAllOwner(){
        return service.getAllOwners();
    }

    @PostMapping("/add")
    public boolean addOwner(@RequestBody OwnerDTO ownerDTO) throws
   ControllerException {
        try {
            return service.addOwner(ownerDTO);
        } catch (ServiceException e) {
            throw new ControllerException("Problem with adding owner",
    e);
        }
    }

    @DeleteMapping("/delete/{id}")
    public boolean removeOwner(@PathVariable("id") String ownerId)
   throws ControllerException {
        try {
            return service.removeOwner(Long.valueOf(ownerId));
        } catch (ServiceException e) {
            throw new ControllerException("Problem with deleting owner
   ", e);
        }
    }
}
```

Листинг 1.24: ControllerException.java

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

Листинг 1.25: Color.java

```
1 package com.griga.dao.colors;
2
3 public enum Color {
4     Black,
5     White,
6     Gray,
7     Orange
8 }
```

Листинг 1.26: Cat.java

```
1  package com.griga.dao.entities;
2
3  import com.griga.dao.colors.Color;
4  import com.griga.dto.CatDTO;
5
6  import javax.persistence.*;
7  import java.sql.Timestamp;
8  import java.util.Objects;
9
10 @Entity
11 @Table(name = "cats", schema = "public", catalog = "postgres")
12 public class Cat {
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     @Id
15     @Basic
16     @Column(name = "id")
17     private Long id;
18     @Basic
19     @Column(name = "name")
20     private String name;
21     @Basic
22     @Column(name = "birthdate")
23     private Timestamp birthdate;
24     @Basic
25     @Column(name = "species")
26     private String species;
27     @Basic
28     @Column(name = "color")
29     @Enumerated(EnumType.STRING)
30     private Color color;
31
32     public Cat() {
33     }
34
35     public Cat(CatDTO catDTO) {
36         name = catDTO.getName();
37         birthdate = catDTO.getBirthdate();
38         species = catDTO.getSpecies();
39         color = catDTO.getColor();
40     }
41
42     public Long getId() {
43         return id;
44     }
45
46     public void setId(Long id) {
47         this.id = id;
48     }
49
```

```
50      public String getName() {
51          return name;
52      }
53
54      public void setName(String name) {
55          this.name = name;
56      }
57
58      public Timestamp getBirthdate() {
59          return birthdate;
60      }
61
62      public void setBirthdate(Timestamp birthdate) {
63          this.birthdate = birthdate;
64      }
65
66      public String getSpecies() {
67          return species;
68      }
69
70      public void setSpecies(String species) {
71          this.species = species;
72      }
73
74      public Color getColor() {
75          return color;
76      }
77
78      public void setColor(Color color) {
79          this.color = color;
80      }
81
82      @Override
83      public boolean equals(Object o) {
84          if (this == o) return true;
85          if (o == null || getClass() != o.getClass()) return false;
86          Cat that = (Cat) o;
87          return Objects.equals(id, that.id) && Objects.equals(name,
        that.name) && Objects.equals(birthdate, that.birthdate) && Objects.
        equals(species, that.species) && Objects.equals(color, that.color);
88      }
89
90      @Override
91      public int hashCode() {
92          return Objects.hash(id, name, birthdate, species, color);
93      }
94
95      @Override
96      public String toString() {
97          return id + ", " + name + ", " + birthdate + ", " + species +
```

```
    ", " + color;
98      }
99  }
```

Листинг 1.27: CatsFriends.java

```
1  package com.griga.dao.entities;
2
3  import com.griga.dto.CatsFriendsDTO;
4
5  import javax.persistence.*;
6  import java.util.Objects;
7
8  @Entity
9  @Table(name = "catsfriends", schema = "public", catalog = "postgres")
10 public class CatsFriends {
11     @GeneratedValue(strategy = GenerationType.IDENTITY)
12     @Id
13     @Column(name = "id")
14     private Long id;
15     @Basic
16     @Column(name = "first_cat_id")
17     private Long firstCatId;
18     @Basic
19     @Column(name = "second_cat_id")
20     private Long secondCatId;
21
22     public CatsFriends() {
23
24     }
25
26     public CatsFriends(CatsFriendsDTO catsFriendsDTO) {
27         firstCatId = catsFriendsDTO.getFirstCatId();
28         secondCatId = catsFriendsDTO.getSecondCatId();
29     }
30
31     public Long getId() {
32         return id;
33     }
34
35     public void setId(Long id) {
36         this.id = id;
37     }
38
39     public long getFirstCatId() {
40         return firstCatId;
41     }
42
43     public void setFirstCatId(Long firstCatId) {
44         this.firstCatId = firstCatId;
45     }
46
47     public Long getSecondCatId() {
48         return secondCatId;
49     }
```

```
50
51      public void setSecondCatId(Long secondCatId) {
52          this.secondCatId = secondCatId;
53      }
54
55      @Override
56      public boolean equals(Object o) {
57          if (this == o) return true;
58          if (o == null || getClass() != o.getClass()) return false;
59          CatsFriends that = (CatsFriends) o;
60          return Objects.equals(id, that.id) && Objects.equals(
        firstCatId, that.firstCatId) && Objects.equals(secondCatId, that.
        secondCatId);
61      }
62
63      @Override
64      public int hashCode() {
65          return Objects.hash(id, firstCatId, secondCatId);
66      }
67  }
```

Листинг 1.28: Owner.java

```
1  package com.griga.dao.entities;
2
3  import com.griga.dto.OwnerDTO;
4
5  import javax.persistence.*;
6  import java.sql.Timestamp;
7  import java.util.Objects;
8
9  @Entity
10 @Table(name = "owner", schema = "public", catalog = "postgres")
11 public class Owner {
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     @Id
14     @Column(name = "id")
15     private Long id;
16     @Basic
17     @Column(name = "name")
18     private String name;
19     @Basic
20     @Column(name = "birthdate")
21     private Timestamp birthdate;
22
23     public Owner() {
24
25     }
26
27     public Owner(OwnerDTO ownerDTO) {
28         name = ownerDTO.getName();
29         birthdate = ownerDTO.getBirthdate();
30     }
31
32     public Long getId() {
33         return id;
34     }
35
36     public void setId(Long id) {
37         this.id = id;
38     }
39
40     public String getName() {
41         return name;
42     }
43
44     public void setName(String name) {
45         this.name = name;
46     }
47
48     public Timestamp getBirthdate() {
49         return birthdate;
```

```
50      }
51
52      public void setBirthdate(Timestamp birthdate) {
53          this.birthdate = birthdate;
54      }
55
56      @Override
57      public boolean equals(Object o) {
58          if (this == o) return true;
59          if (o == null || getClass() != o.getClass()) return false;
60          Owner that = (Owner) o;
61          return Objects.equals(id, that.id) && Objects.equals(name,
    that.name) && Objects.equals(birthdate, that.birthdate);
62      }
63
64      @Override
65      public int hashCode() {
66          return Objects.hash(id, name, birthdate);
67      }
68
69      @Override
70      public String toString() {
71          return id + ", " + name + ", " + birthdate;
72      }
73  }
```

Листинг 1.29: OwnersCats.java

```
package com.griga.dao.entities;

import com.griga.dto.OwnersCatsDTO;

import javax.persistence.*;
import java.util.Objects;

@Entity
@Table(name = "ownerscats", schema = "public", catalog = "postgres")
public class OwnersCats {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id")
    private Long id;
    @Basic
    @Column(name = "owner_id")
    private Long ownerId;
    @Basic
    @Column(name = "cat_id")
    private Long catId;

    public OwnersCats() {

    }

    public OwnersCats(OwnersCatsDTO ownersCatsDTO) {
        ownerId = ownersCatsDTO.getOwnerId();
        catId = ownersCatsDTO.getCatId();
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Long getOwnerId() {
        return ownerId;
    }

    public void setOwnerId(Long ownerId) {
        this.ownerId = ownerId;
    }

    public Long getCatId() {
        return catId;
    }
```

```
50
51      public void setCatId(Long catId) {
52          this.catId = catId;
53      }
54
55      @Override
56      public boolean equals(Object o) {
57          if (this == o) return true;
58          if (o == null || getClass() != o.getClass()) return false;
59          OwnersCats that = (OwnersCats) o;
60          return Objects.equals(id, that.id) && Objects.equals(ownerId,
        that.ownerId) && Objects.equals(catId, that.catId);
61      }
62
63      @Override
64      public int hashCode() {
65          return Objects.hash(id, ownerId, catId);
66      }
67  }
```

Листинг 1.30: CatRepository.java

```
1  package com.griga.dao.implementations;
2
3  import com.griga.dao.entities.Cat;
4  import org.springframework.data.jpa.repository.JpaRepository;
5  import org.springframework.stereotype.Repository;
6
7  @Repository
8  public interface CatRepository extends JpaRepository<Cat, Long> {
9
10 }
```

Листинг 1.31: CatsFriendsRepository.java

```
1  package com.griga.dao.implementations;
2
3  import com.griga.dao.entities.CatsFriends;
4
5  import org.springframework.data.jpa.repository.JpaRepository;
6  import org.springframework.stereotype.Repository;
7
8  @Repository
9  public interface CatsFriendsRepository extends JpaRepository<
       CatsFriends, Long> {
10
11 }
```

Листинг 1.32: OwnerRepository.java

```
1  package com.griga.dao.implementations;
2
3  import com.griga.dao.entities.Owner;
4  import org.springframework.data.jpa.repository.JpaRepository;
5  import org.springframework.stereotype.Repository;
6
7
8  @Repository
9  public interface OwnerRepository extends JpaRepository<Owner, Long> {
10
11 }
```

Листинг 1.33: OwnersCatsRepository.java

```
1  package com.griga.dao.implementations;
2
3  import com.griga.dao.entities.OwnersCats;
4  import org.springframework.data.jpa.repository.JpaRepository;
5  import org.springframework.stereotype.Repository;
6
7
8  @Repository
9  public interface OwnersCatsRepository extends JpaRepository<OwnersCats
       , Long> {
10
11 }
```

Листинг 1.34: DAOException.java

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

Листинг 1.35: CatDTO.java

```
package com.griga.dto;

import com.griga.dao.colors.Color;
import com.griga.dao.entities.Cat;

import java.sql.Timestamp;

public class CatDTO {
    private Long id;
    private String name;
    private Timestamp birthdate;
    private String species;
    private Color color;

    public CatDTO() {

    }

    public CatDTO(Cat cat) {
        id = cat.getId();
        name = cat.getName();
        birthdate = cat.getBirthdate();
        species = cat.getSpecies();
        color = cat.getColor();
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Timestamp getBirthdate() {
        return birthdate;
    }

    public void setBirthdate(Timestamp birthdate) {
        this.birthdate = birthdate;
    }
```

```
50
51      public String getSpecies() {
52          return species;
53      }
54
55      public void setSpecies(String species) {
56          this.species = species;
57      }
58
59      public Color getColor() {
60          return color;
61      }
62
63      public void setColor(Color color) {
64          this.color = color;
65      }
66  }
```

Листинг 1.36: CatsFriendsDTO.java

```
1  package com.griga.dto;
2
3  import com.griga.dao.entities.CatsFriends;
4
5  public class CatsFriendsDTO {
6      private Long id;
7      private Long firstCatId;
8      private Long secondCatId;
9
10     public CatsFriendsDTO() {
11
12     }
13
14     public CatsFriendsDTO(CatsFriends catsFriends) {
15         id = catsFriends.getId();
16         firstCatId = catsFriends.getFirstCatId();
17         secondCatId = catsFriends.getSecondCatId();
18     }
19
20     public Long getId() {
21         return id;
22     }
23
24     public void setId(Long id) {
25         this.id = id;
26     }
27
28     public Long getFirstCatId() {
29         return firstCatId;
30     }
31
32     public void setFirstCatId(Long firstCatId) {
33         this.firstCatId = firstCatId;
34     }
35
36     public Long getSecondCatId() {
37         return secondCatId;
38     }
39
40     public void setSecondCatId(Long secondCatId) {
41         this.secondCatId = secondCatId;
42     }
43 }
```

Листинг 1.37: OwnerDTO.java

```
1  package com.griga.dto;
2
3  import com.griga.dao.entities.Owner;
4
5  import java.sql.Timestamp;
6
7  public class OwnerDTO {
8      private Long id;
9      private String name;
10     private Timestamp birthdate;
11
12     public OwnerDTO() {
13
14     }
15
16     public OwnerDTO(Owner owner) {
17         id = owner.getId();
18         name = owner.getName();
19         birthdate = owner.getBirthdate();
20     }
21
22     public Long getId() {
23         return id;
24     }
25
26     public void setId(Long id) {
27         this.id = id;
28     }
29
30     public String getName() {
31         return name;
32     }
33
34     public void setName(String name) {
35         this.name = name;
36     }
37
38     public Timestamp getBirthdate() {
39         return birthdate;
40     }
41
42     public void setBirthdate(Timestamp birthdate) {
43         this.birthdate = birthdate;
44     }
45 }
```

Листинг 1.38: OwnersCatsDTO.java

```
1  package com.griga.dto;
2
3  import com.griga.dao.entities.OwnersCats;
4
5  public class OwnersCatsDTO {
6      private Long id;
7      private Long ownerId;
8      private Long catId;
9
10     public OwnersCatsDTO() {
11
12     }
13
14     public OwnersCatsDTO(OwnersCats ownersCats) {
15         id = ownersCats.getId();
16         ownerId = ownersCats.getOwnerId();
17         catId = ownersCats.getCatId();
18     }
19
20     public Long getId() {
21         return id;
22     }
23
24     public void setId(Long id) {
25         this.id = id;
26     }
27
28     public Long getOwnerId() {
29         return ownerId;
30     }
31
32     public void setOwnerId(Long ownerId) {
33         this.ownerId = ownerId;
34     }
35
36     public Long getCatId() {
37         return catId;
38     }
39
40     public void setCatId(Long catId) {
41         this.catId = catId;
42     }
43 }
```

Листинг 1.39: SpringService.java

```
1  package com.griga.service;
2
3  import com.griga.dao.colors.Color;
4  import com.griga.dao.entities.Cat;
5  import com.griga.dao.entities.CatsFriends;
6  import com.griga.dao.entities.Owner;
7  import com.griga.dao.entities.OwnersCats;
8  import com.griga.dao.implementations.CatRepository;
9  import com.griga.dao.implementations.CatsFriendsRepository;
10 import com.griga.dao.implementations.OwnerRepository;
11 import com.griga.dao.implementations.OwnersCatsRepository;
12 import com.griga.dto.CatDTO;
13 import com.griga.dto.CatsFriendsDTO;
14 import com.griga.dto.OwnerDTO;
15 import com.griga.dto.OwnersCatsDTO;
16 import com.griga.service.tools.ServiceException;
17 import org.springframework.beans.factory.annotation.Autowired;
18 import org.springframework.stereotype.Service;
19
20 import java.util.List;
21 import java.util.Objects;
22
23 @Service
24 public class SpringService {
25
26     @Autowired
27     private OwnerRepository owner;
28     @Autowired
29     private CatRepository cat;
30     @Autowired
31     private OwnersCatsRepository ownersCats;
32     @Autowired
33     private CatsFriendsRepository catsFriends;
34
35     public boolean addCat(CatDTO catDTO) throws ServiceException {
36         if (catDTO == null) throw new ServiceException("Cat can not be
       null!");
37
38         cat.save(new Cat(catDTO));
39         return true;
40     }
41
42     public boolean removeCat(Long catId) throws ServiceException {
43         List<CatsFriends> catsAndFriends = catsFriends.findAll();
44         for (CatsFriends catAndFriend: catsAndFriends) {
45             if (catAndFriend.getFirstCatId() == catId || Objects.
       equals(catAndFriend.getSecondCatId(), catId)) {
46                 catsFriends.delete(catsFriends.getById(catAndFriend.
       getId()));
```

```
47              }
48          }
49
50          List<OwnersCats> ownersAndCats = ownersCats.findAll();
51          for (OwnersCats ownerAndCat: ownersAndCats) {
52              if (Objects.equals(ownerAndCat.getCatId(), catId)) {
53                  ownersCats.delete(ownersCats.getById(ownerAndCat.getId
    ()));
54              }
55          }
56
57          cat.delete(cat.getById(catId));
58
59          return true;
60      }
61
62      public boolean addOwner(OwnerDTO ownerDTO) throws ServiceException
        {
63          if (ownerDTO == null) throw new ServiceException("Owner can
    not be null!");
64
65          owner.save(new Owner(ownerDTO));
66          return true;
67      }
68
69      public boolean removeOwner(Long ownerId) throws ServiceException {
70          List<OwnersCats> ownersAndCats = ownersCats.findAll();
71          for (OwnersCats ownerAndCat: ownersAndCats) {
72              if (Objects.equals(ownerAndCat.getOwnerId(), ownerId)) {
73                  ownersCats.delete(ownersCats.getById(ownerAndCat.getId
    ()));
74              }
75          }
76
77          owner.delete(owner.getById(ownerId));
78
79          return true;
80      }
81
82      public boolean addCatToOwner(CatDTO catDTO, Long ownerId) throws
    ServiceException {
83          if (catDTO == null) throw new ServiceException("Cat can not be
     null!");
84
85          List<OwnersCats> ownersAndCats = ownersCats.findAll();
86          for (OwnersCats ownerAndCat: ownersAndCats) {
87              if (Objects.equals(ownerAndCat.getOwnerId(), ownerId) &&
    Objects.equals(ownerAndCat.getCatId(), catDTO.getId())) {
88                  throw new ServiceException("Such owner-cat pair
    already exist!");
```

```
89              }
90          }
91
92          OwnersCatsDTO ownersCatsDTO = new OwnersCatsDTO();
93          ownersCatsDTO.setOwnerId(ownerId);
94          ownersCatsDTO.setCatId(catDTO.getId());
95
96          ownersCats.save(new OwnersCats(ownersCatsDTO));
97
98          return true;
99      }
100
101     public boolean removeCatFromOwner(CatDTO catDTO, Long ownerId)
    throws ServiceException {
102         if (catDTO == null) throw new ServiceException("Cat can not be
     null!");
103
104         List<OwnersCats> ownersAndCats = ownersCats.findAll();
105         for (OwnersCats ownerAndCat: ownersAndCats) {
106             if (Objects.equals(ownerAndCat.getOwnerId(), ownerId) &&
    Objects.equals(ownerAndCat.getCatId(), catDTO.getId())) {
107                 ownersCats.delete(ownersCats.getById(ownerAndCat.getId
    ()));
108             }
109         }
110
111         return true;
112     }
113
114     public boolean makeFriendship(CatDTO firstCat, Long secondCatId)
    throws ServiceException {
115         if (firstCat == null) throw new ServiceException("Cat can not
    be null!");
116
117         CatsFriendsDTO catsFriendsDTO = new CatsFriendsDTO();
118         catsFriendsDTO.setFirstCatId(firstCat.getId());
119         catsFriendsDTO.setSecondCatId(secondCatId);
120
121         catsFriends.save(new CatsFriends(catsFriendsDTO));
122
123         return true;
124     }
125
126     public boolean breakFriendship(CatDTO firstCat, Long secondCatId)
    throws ServiceException {
127         if (firstCat == null) throw new ServiceException("Cat can not
    be null!");
128
129         List<CatsFriends> catsAndFriends = catsFriends.findAll();
130         for (CatsFriends catAndFriend: catsAndFriends) {
```

```
131             if (catAndFriend.getFirstCatId() == firstCat.getId() &&
    Objects.equals(catAndFriend.getSecondCatId(), secondCatId) ||
132                     catAndFriend.getFirstCatId() == secondCatId &&
    Objects.equals(catAndFriend.getSecondCatId(), firstCat.getId())) {
133                 catsFriends.delete(catAndFriend);
134             }
135         }
136
137         return true;
138     }
139
140     public List<CatDTO> getCatsByColor(Color color) {
141         return getAllCats().stream().filter(cat -> Objects.equals(cat.
    getColor(), color)).toList();
142     }
143
144     public List<CatDTO> getAllCats() {
145         return cat.findAll().stream().map(CatDTO::new).toList();
146     }
147
148     public List<OwnerDTO> getAllOwners() {
149         return owner.findAll().stream().map(OwnerDTO::new).toList();
150     }
151
152 }
```

Листинг 1.40: ServiceException.java

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

Листинг 1.41: CatControllerTest.java

```
1  package com.griga.controller.controllers;
2
3  import com.griga.service.SpringService;
4  import org.junit.jupiter.api.Test;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.boot.test.autoconfigure.web.servlet.
       WebMvcTest;
7  import org.springframework.boot.test.mock.mockito.MockBean;
8  import org.springframework.test.web.servlet.MockMvc;
9
10 import static org.springframework.test.web.servlet.request.
       MockMvcRequestBuilders.get;
11 import static org.springframework.test.web.servlet.result.
       MockMvcResultMatchers.status;
12
13 @WebMvcTest(CatController.class)
14 class CatControllerTest {
15     @MockBean
16     SpringService service;
17     @Autowired
18     private MockMvc mockMvc;
19
20     @Test
21     void getByColor_ThrowsException() throws Exception {
22         String url = "http://localhost:8080/cat/get-by-color/Red";
23         mockMvc.perform(get(url)).andExpect(status().is4xxClientError
       ());
24     }
25 }
```