Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Лабороторная работа №4

Выполнил студент:

Карепин Денис Дмитриевич
группа: M32071

Проверил:

Чикишев Константин Максимович

Санкт-Петербург,
2022 г.

1.1. Текст задания

4 лабораторная
Владельцы недовольны, что информацию о котиках может получить кто угодно. В этой лабораторной мы добавим авторизацию к сервису.
Добавляется роль администратора. Он имеет доступ ко всем методам и может создавать новых пользователей. Пользователь связан с владельцем в соотношении 1:1.
Методы по получению информации и котиках и владельцах должны быть защищены Spring Security. Доступ к соответствующим endpoint'ам имеют только владельцы котиков и администраторы. Доступ к методам для фильтрации имеют все авторизованные пользователи, но на выходе получают только данные о своих котиках.
Внимание: эндпоинты, созданные на предыдущем этапе, не должны быть удалены.

1.2. Решение

Листинг 1.1: mainBank.java

```
1  import tools.CentralBankException;
2  import tools.ConsoleInterface;
3
4  public class mainBank {
5      public static void main(String[] args) throws CentralBankException
       {
6          var cons = new ConsoleInterface();
7          ConsoleInterface.input();
8
9      }
10 }
```

Листинг 1.2: AccountOption.java

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

Листинг 1.3: CreditAccount.java

```
1  package accountServices;
2
3  import banksServices.Bank;
4  import clientServices.Client;
5  import tools.CentralBankException;
6
7  import java.util.UUID;
8
9  public class CreditAccount implements IAccount {
10     private double balance;
11     private double commissionUsing;
12     private double creditLimit;
13     private String numberOfAccount;
14     private boolean verification;
15     public Bank belongBank;
16
17     public CreditAccount(Client user, Bank bank, double amount) throws
       CentralBankException {
18         if (bank == null) {
19             throw new CentralBankException("null bank");
20         }
21         if (user == null) {
22             throw new CentralBankException("null client");
23         }
24         verification = user.getVerification();
25         commissionUsing = bank.getCommissionUsingForCreditAccounts();
26         creditLimit = bank.getCreditLimitForCreditAccounts();
27         balance = amount;
28         belongBank = bank;
29         numberOfAccount = UUID.randomUUID().toString();
30     }
31
32     public void withdrawalMoney(double amount) {
33         if (balance - amount > -creditLimit) {
34             balance -= amount;
35         }
36     }
37
38     public void replenishmentMoney(double amount) {
39         balance += amount;
40     }
41
42     public void transferMoney(IAccount account, double amount) {
43         withdrawalMoney(amount);
44         account.replenishmentMoney(amount);
45     }
46
47     public void actionWithAccount() {
48         if ((balance < 0) && (balance - commissionUsing >= -
```

```
    creditLimit)) {
49              balance -= commissionUsing;
50          }
51      }
52
53      public String getIdAccount() {
54          return numberOfAccount;
55      }
56
57      public boolean checkVerification() {
58          return verification;
59      }
60
61      public Bank getBelongBank() {
62          return belongBank;
63      }
64  }
```

Листинг 1.4: DebitAccount.java

```
1  package accountServices;
2
3  import banksServices.Bank;
4  import clientServices.Client;
5  import tools.CentralBankException;
6
7  import java.util.UUID;
8
9  public class DebitAccount implements IAccount {
10     private double balance;
11     private double percentageOnBalance;
12     private String numberOfAccount;
13     private boolean verification;
14
15     public DebitAccount(Client user, Bank bank, double amount) throws
       CentralBankException {
16         if (bank == null) {
17             throw new CentralBankException("null bank");
18         }
19         if (user == null) {
20             throw new CentralBankException("null client");
21         }
22         verification = user.getVerification();
23         percentageOnBalance = bank.
       getPercentageOnBalanceForDebitAccounts();
24         BelongBank = bank;
25         balance = amount;
26         numberOfAccount = UUID.randomUUID().toString();
27     }
28
29     public Bank BelongBank;
30
31     public void withdrawalMoney(double amount) {
32         balance -= amount;
33     }
34
35     public void replenishmentMoney(double amount) {
36         balance += amount;
37     }
38
39     public void transferMoney(IAccount account, double amount) {
40         withdrawalMoney(amount);
41         account.replenishmentMoney(amount);
42     }
43
44     public void actionWithAccount() {
45         balance += balance * percentageOnBalance / 100;
46     }
47
```

```
48      public String getIdAccount() {
49          return numberOfAccount;
50      }
51
52      public boolean checkVerification() {
53          return verification;
54      }
55
56      public Bank getBelongBank() {
57          return BelongBank;
58      }
59  }
```

Листинг 1.5: DepositAccount.java

```
1  package accountServices;
2
3  import banksServices.Bank;
4  import clientServices.Client;
5  import tools.CentralBankException;
6  import tools.Pair;
7
8  import java.util.UUID;
9
10 public class DepositAccount implements IAccount {
11     private double balance;
12     private double percentage;
13     private String numberOfAccount;
14     private boolean verification;
15
16     public DepositAccount(Client user, Bank bank, double amount)
       throws CentralBankException {
17         if (bank == null) {
18             throw new CentralBankException("null bank");
19         }
20         if (user == null) {
21             throw new CentralBankException("null client");
22         }
23         verification = user.getVerification();
24         percentage = bank.getPercentageOnBalanceForDepositAccounts().
25                 getPairsSumAndPercent().stream().filter(x -> x.getSum
       () > amount).findFirst().orElse(new Pair(0.0, 0.0)).getPercentage()
       ;
26         balance = amount;
27         BelongBank = bank;
28         numberOfAccount = UUID.randomUUID().toString();
29     }
30
31     public Bank BelongBank;
32
33     public void withdrawalMoney(double amount) {
34         balance -= amount;
35     }
36
37     public void replenishmentMoney(double amount) {
38         balance += amount;
39     }
40
41     public void transferMoney(IAccount account, double amount) {
42     }
43
44     public void actionWithAccount() {
45         balance += balance * percentage / 100;
46     }
```

```
47
48      public String getIdAccount() {
49          return numberOfAccount;
50      }
51
52      public boolean checkVerification() {
53          return verification;
54      }
55
56      public Bank getBelongBank() {
57          return BelongBank;
58      }
59  }
```

Листинг 1.6: DepositAccountPercentage.java

```
package accountServices;

import tools.Pair;

import java.util.ArrayList;

public class DepositAccountPercentage {
    private ArrayList<Pair> pairsSumAndPercent;

    public DepositAccountPercentage() {
        pairsSumAndPercent = new ArrayList<Pair>();
    }

    public void addParametersForDepositAccountBank(double sum, double
    percentage) {
        pairsSumAndPercent.add(new Pair(sum, percentage));
    }

    public ArrayList<Pair> getPairsSumAndPercent() {
        return pairsSumAndPercent;
    }

    public void setPairsSumAndPercent(ArrayList<Pair> value) {
        this.pairsSumAndPercent = value;
    }

}
```

Листинг 1.7: IAccount.java

```
1  package accountServices;
2
3  import banksServices.Bank;
4
5  public interface IAccount {
6      void withdrawalMoney(double amount);
7
8      void replenishmentMoney(double amount);
9
10     void transferMoney(IAccount account, double amount);
11
12     void actionWithAccount();
13
14     String getIdAccount();
15
16     boolean checkVerification();
17
18     Bank getBelongBank();
19 }
```

Листинг 1.8: Bank.java

```
1  package banksServices;
2
3  import accountServices.DepositAccountPercentage;
4  import accountServices.IAccount;
5  import clientServices.Client;
6  import tools.CentralBankException;
7  import tools.EventRegistrar;
8
9  import java.util.*;
10
11 public class Bank {
12     public EventRegistrar events;
13     private HashMap<Client, ArrayList<IAccount>> baseBank;
14     private ArrayList<Transaction> transactions;
15     private double limitForNotVerification;
16     private double creditLimitForCreditAccounts;
17     private double commissionUsingForCreditAccounts;
18     private double percentageOnBalanceForDebitAccounts;
19     private String name;
20     private DepositAccountPercentage
       percentageOnBalanceForDepositAccounts;
21
22     public Bank(String name, double limitForNotVerification, double
       creditLimitForCreditAccounts, double
       commissionUsingForCreditAccounts,
23                 DepositAccountPercentage
       percentageOnBalanceForDepositAccounts, double
       percentageOnBalanceForDebitAccounts) throws CentralBankException {
24         if (name == null) throw new CentralBankException("Incorrect
       name");
25         this.name = name;
26         this.limitForNotVerification = limitForNotVerification;
27         this.creditLimitForCreditAccounts =
       creditLimitForCreditAccounts;
28         this.commissionUsingForCreditAccounts =
       commissionUsingForCreditAccounts;
29         this.percentageOnBalanceForDepositAccounts =
       percentageOnBalanceForDepositAccounts;
30         this.percentageOnBalanceForDebitAccounts =
       percentageOnBalanceForDebitAccounts;
31         this.baseBank = new HashMap<>();
32         this.transactions = new ArrayList<>();
33         this.events = new EventRegistrar("Change name",
34                 "Change creditLimitForCreditAccounts",
35                 "Change commissionUsingForCreditAccounts",
36                 "Change limitForNotVerification",
37                 "Change percentageOnBalanceForDebitAccounts",
38                 "Change percentageOnBalanceForDepositAccounts");
39     }
```

```
40
41      // public delegate void ChangeFieldInBanks(double other);
42      // public event ChangeFieldInBanks ChangeFieldInBank;
43
44      public String getName() {
45          return name;
46      }
47
48      public void setName(String name) {
49          events.notify("Change name");
50          this.name = name;
51      }
52
53      public double getCreditLimitForCreditAccounts() {
54          return creditLimitForCreditAccounts;
55      }
56
57      public void setCreditLimitForCreditAccounts(double value) {
58          events.notify("Change creditLimitForCreditAccounts");
59          this.creditLimitForCreditAccounts = value;
60      }
61
62      public double getCommissionUsingForCreditAccounts() {
63          return commissionUsingForCreditAccounts;
64      }
65
66      public void setCommissionUsingForCreditAccounts(double value) {
67          events.notify("Change commissionUsingForCreditAccounts");
68          this.commissionUsingForCreditAccounts = value;
69      }
70
71      public double getLimitForNotVerification() {
72          return limitForNotVerification;
73      }
74
75      public void setLimitForNotVerification(double value) {
76          events.notify("Change limitForNotVerification");
77          this.limitForNotVerification = value;
78      }
79
80      public double getPercentageOnBalanceForDebitAccounts() {
81          return percentageOnBalanceForDebitAccounts;
82      }
83
84      public void setPercentageOnBalanceForDebitAccounts(double value) {
85          events.notify("Change percentageOnBalanceForDebitAccounts");
86          this.percentageOnBalanceForDebitAccounts = value;
87      }
88
89      public DepositAccountPercentage
```

```
    getPercentageOnBalanceForDepositAccounts() {
90          return percentageOnBalanceForDepositAccounts;
91      }
92
93      public void setPercentageOnBalanceForDepositAccounts(
    DepositAccountPercentage value) {
94          events.notify("Change percentageOnBalanceForDepositAccounts");
95          this.percentageOnBalanceForDepositAccounts = value;
96      }
97
98      public void registerClient(Client client, IAccount account) throws
     CentralBankException {
99          if (client == null) throw new CentralBankException("Incorrect
    client");
100         if (account == null) throw new CentralBankException("Incorrect
     account");
101         if (baseBank.containsKey(client))
102             baseBank.get(client).add(account);
103         else
104             baseBank.put(client, new ArrayList<IAccount>(List.of(
    account)));
105         client.createAccount(this, getInfoAccounts(client));
106     }
107
108     public IAccount findAccount(String numberId) throws
    CentralBankException {
109         if (numberId == null) throw new CentralBankException("
    Incorrect numberId");
110         return baseBank.values()
111                 .stream()
112                 .flatMap(Collection::stream)
113                 .filter(i -> Objects.equals(i.getIdAccount(), numberId
    ))
114                 .findFirst()
115                 .orElse(null);
116     }
117
118     public void accruePercentage() {
119         baseBank.values().stream().flatMap(Collection::stream).forEach
    (IAccount::actionWithAccount);
120     }
121
122     public void addTransaction(Transaction transaction) {
123         transactions.add(transaction);
124     }
125
126     private ArrayList<IAccount> getInfoAccounts(Client client) {
127         return baseBank.getOrDefault(client, null);
128     }
129 }
```

Листинг 1.9: CentralBank.java

```
1  package banksServices;
2
3  import accountServices.*;
4  import clientServices.Client;
5  import tools.CentralBankException;
6
7  import java.util.ArrayList;
8  import java.util.List;
9  import java.util.Objects;
10
11 public class CentralBank {
12     private List<Bank> banks;
13     private List<Transaction> transactions;
14
15     public CentralBank() {
16         banks = new ArrayList<Bank>();
17         transactions = new ArrayList<Transaction>();
18     }
19
20     public Bank addBankToBase(String name, double
       limitForNotVerification, double creditLimitForCreditAccounts,
       double commissionUsingForCreditAccounts, DepositAccountPercentage
       percentageOnBalanceForDepositAccounts, double
       percentageOnBalanceForDebitAccounts) throws CentralBankException {
21         banks.add(new Bank(name, limitForNotVerification,
       creditLimitForCreditAccounts, commissionUsingForCreditAccounts,
       percentageOnBalanceForDepositAccounts,
       percentageOnBalanceForDebitAccounts));
22         return banks.get(banks.size() - 1);
23     }
24
25     public IAccount regAccountClientInBank(Bank bank, Client client,
       AccountOption option, double amount) throws CentralBankException {
26         if (bank == null) {
27             throw new CentralBankException("null bank");
28         }
29         if (client == null) {
30             throw new CentralBankException("null client");
31         }
32         if (!banks.contains(bank)) {
33             throw new CentralBankException("Bank dont registered");
34         }
35         if (amount < 0) {
36             throw new CentralBankException("Negative balance");
37         }
38         IAccount account;
39         switch (option) {
40             case Credit -> {
41                 account = new CreditAccount(client, bank, amount);
```

```
42                  bank.registerClient(client, account);
43                  return account;
44              }
45              case Deposit -> {
46                  account = new DepositAccount(client, bank, amount);
47                  bank.registerClient(client, account);
48                  return account;
49              }
50              case Debit -> {
51                  account = new DebitAccount(client, bank, amount);
52                  bank.registerClient(client, account);
53                  return account;
54              }
55              default -> throw new CentralBankException("{option} -
    Incorrect options");
56          }
57      }
58
59      public Transaction withdrawalMoney(IAccount account, double amount
    ) throws CentralBankException {
60          if (!account.checkVerification()
61                  &&
62                  account.getBelongBank().getLimitForNotVerification() <
     amount) {
63              throw new CentralBankException("Attempt to withdraw money
    from an unverified account");
64          }
65          var tmpTransaction = new Transaction(account.getIdAccount(),
    null, amount);
66          transactions.add(tmpTransaction);
67          account.getBelongBank().addTransaction(tmpTransaction);
68          account.withdrawalMoney(amount);
69          return transactions.get(transactions.size() - 1);
70      }
71
72      public Transaction replenishmentMoney(IAccount account, double
    amount) {
73          var tmpTransaction = new Transaction(null, account.
    getIdAccount(), amount);
74          transactions.add(tmpTransaction);
75          account.getBelongBank().addTransaction(tmpTransaction);
76          account.replenishmentMoney(amount);
77          return transactions.get(transactions.size() - 1);
78      }
79
80      public Transaction transferMoney(IAccount account1, IAccount
    account2, double amount) throws CentralBankException {
81          if (!account1.checkVerification()
82                  &&
83                  account1.getBelongBank().getLimitForNotVerification()
```

```
< amount) {
84              throw new CentralBankException("Attempt to withdraw money
    from an unverified account");
85          }
86          var tmpTransaction = new Transaction(account1.getIdAccount(),
    account2.getIdAccount(), amount);
87          transactions.add(tmpTransaction);
88          account1.getBelongBank().addTransaction(tmpTransaction);
89          account2.getBelongBank().addTransaction(tmpTransaction);
90          account1.transferMoney(account2, amount);
91          return transactions.get(transactions.size() - 1);
92      }
93
94      public void cancelTransaction(Transaction transaction) throws
    CentralBankException {
95          if (transaction == null) {
96              throw new CentralBankException("Incorrect transaction");
97          }
98          IAccount tmpTransferAccount = null;
99          IAccount tmpWithdrawalAccount = null;
100         if (transaction.getTransferAccount() != null && transaction.
    getWithdrawalAccount() != null) {
101             for (Bank bank : banks) {
102                 tmpTransferAccount = bank.findAccount(transaction.
    getTransferAccount());
103                 tmpWithdrawalAccount = bank.findAccount(transaction.
    getWithdrawalAccount());
104                 if (tmpTransferAccount != null && tmpWithdrawalAccount
     != null) break;
105             }
106
107             Objects.requireNonNull(tmpTransferAccount).
    replenishmentMoney(transaction.getAmount());
108             Objects.requireNonNull(tmpWithdrawalAccount).
    withdrawalMoney(transaction.getAmount());
109         } else if (transaction.getWithdrawalAccount() == null &&
    transaction.getTransferAccount() != null) {
110             for (Bank bank : banks) {
111                 tmpTransferAccount = bank.findAccount(transaction.
    getTransferAccount());
112                 if (tmpTransferAccount != null) break;
113             }
114
115             Objects.requireNonNull(tmpTransferAccount).withdrawalMoney
    (transaction.getAmount());
116         } else if (transaction.getTransferAccount() == null &&
    transaction.getWithdrawalAccount() != null) {
117             for (Bank bank : banks) {
118                 tmpWithdrawalAccount = bank.findAccount(transaction.
    getWithdrawalAccount());
```

```
119                 if (tmpWithdrawalAccount != null) break;
120             }
121
122             Objects.requireNonNull(tmpWithdrawalAccount).
    replenishmentMoney(transaction.getAmount());
123         }
124
125         transactions.remove(transaction);
126     }
127
128     public void manageTime(int countOfDay) {
129         for (int i = 0; i < countOfDay % 30; i++) {
130             for (Bank bank : banks) {
131                 bank.accruePercentage();
132             }
133         }
134     }
135
136     public boolean findBank(Bank bank) {
137         return banks.contains(bank);
138     }
139 }
```

Листинг 1.10: Transaction.java

```
package banksServices;

public class Transaction {
    private String withdrawalAccount;
    private String transferAccount;
    private double amount;

    public Transaction(String withdrawalAccount, String
   transferAccount, double amount) {
        this.withdrawalAccount = withdrawalAccount;
        this.transferAccount = transferAccount;
        this.amount = amount;
    }

    public String getWithdrawalAccount() {
        return withdrawalAccount;
    }

    public void setWithdrawalAccount(String value) {
        this.withdrawalAccount = value;
    }

    public String getTransferAccount() {
        return transferAccount;
    }

    public void setTransferAccount(String value) {
        this.transferAccount = value;
    }

    public double getAmount() {
        return amount;
    }

    public void setAmount(double value) {
        this.amount = value;
    }

    public String toString() {
        return this.withdrawalAccount + " to " + this.transferAccount
   + "of" + this.amount;
    }
}
```

Листинг 1.11: Client.java

```
1  package clientServices;
2
3  import accountServices.IAccount;
4  import banksServices.Bank;
5  import tools.CentralBankException;
6
7  import java.util.ArrayList;
8  import java.util.HashMap;
9  import java.util.UUID;
10
11
12 public class Client {
13     private HashMap<Bank, ArrayList<IAccount>>
       clientCollectionAccounts;
14     private UUID id;
15     private String name;
16     private String surname;
17     private String address;
18     private String passport;
19     private boolean isAllInfo;
20
21     public Client(String name, String surname, String address, String
       passport) throws CentralBankException {
22         this.id = UUID.randomUUID();
23         if (name.isBlank()) {
24             throw new CentralBankException("Incorrect name");
25         }
26         this.name = name;
27         if (surname.isBlank()) {
28             throw new CentralBankException("Incorrect surname");
29         }
30         this.surname = surname;
31         this.address = address;
32         this.passport = passport;
33         this.clientCollectionAccounts = new HashMap<Bank, ArrayList<
       IAccount>>();
34         isAllInfo = this.address != null && this.passport != null && !
       this.address.isBlank() && !this.passport.isBlank();
35     }
36
37     public static void update(String other) {
38         System.out.print(other);
39     }
40
41     public static ClientBuilder Builder(String name, String surname)
       throws CentralBankException {
42         return new ClientBuilder().addName(name).addSurname(surname);
43     }
44
```

```
45      public boolean getVerification() {
46          return isAllInfo;
47      }
48
49
50      public void createAccount(Bank bank, ArrayList<IAccount> accounts)
        {
51          if (!clientCollectionAccounts.containsKey(bank))
52              clientCollectionAccounts.put(bank, accounts);
53          else {
54              clientCollectionAccounts.get(bank).addAll(accounts);
55          }
56      }
57  }
```

Листинг 1.12: ClientBuilder.java

```
1  package clientServices;
2
3  import tools.CentralBankException;
4
5  public class ClientBuilder {
6      private String name;
7      private String surname;
8      private String address;
9      private String passport;
10
11     public ClientBuilder addName(String name) throws
       CentralBankException {
12         if ((name == null) || (name.isBlank())) {
13             throw new CentralBankException("Incorrect name");
14         }
15         this.name = name;
16         return this;
17     }
18
19     public ClientBuilder addSurname(String surname) throws
       CentralBankException {
20         if ((surname == null) || (surname.isBlank())) {
21             throw new CentralBankException("Incorrect surname");
22         }
23         this.surname = surname;
24         return this;
25     }
26
27     public ClientBuilder addAddress(String address) throws
       CentralBankException {
28         if ((address == null) || (address.isBlank())) {
29             throw new CentralBankException("Incorrect address");
30         }
31         this.address = address;
32
33         return this;
34     }
35
36     public ClientBuilder addPassport(String passport) throws
       CentralBankException {
37         if ((passport == null) || (passport.isBlank())) {
38             throw new CentralBankException("Incorrect passport");
39         }
40         this.passport = passport;
41         return this;
42     }
43
44     public Client getClient() throws CentralBankException {
45         return new Client(name, surname, address, passport);
```

```
46          }
47  }
```

Листинг 1.13: CentralBankException.java

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

Листинг 1.14: ConsoleInterface.java

```
package tools;

import accountServices.AccountOption;
import accountServices.DepositAccountPercentage;
import accountServices.IAccount;
import banksServices.Bank;
import banksServices.CentralBank;
import banksServices.Transaction;
import clientServices.Client;

import java.util.Scanner;

public class ConsoleInterface {
    public static void input() throws CentralBankException {
        var centralBank = new CentralBank();
        Client client = null;
        IAccount account1 = null;
        IAccount account2 = null;
        Bank bank = null;
        Transaction transaction = null;
        String b = null;
        while (b != "Q") {
            System.out.println("1 - Start Create Bank");
            System.out.println("2 - Start Create Client");
            System.out.println("3 - Start Create Account");
            System.out.println("4 - Start Do Transaction");
            System.out.println("5 - Start Canceled Transaction");
            System.out.println("Q - Exit Program");
            Scanner in = new Scanner(System.in);
            b = in.nextLine();

            switch (b) {
                case "1": {
                    System.out.println("Set Name\nExample: string");
                    String name = in.nextLine();
                    System.out.println("Set limitForNotVerification\
nExample: double");
                    double limitForNotVerification = in.nextDouble();
                    System.out.println("Set
creditLimitForCreditAccounts\nExample: double");
                    double creditLimitForCreditAccounts = in.
nextDouble();
                    System.out.println("Set
commissionUsingForCreditAccounts\nExample: double");
                    double commissionUsingForCreditAccounts = in.
nextDouble();
                    System.out.println("Set
percentageOnBalanceForDepositAccounts\nExample: 50000 3\nExample:
100000 5\nExample:1000000 6");
```

```
43                  DepositAccountPercentage
    percentageOnBalanceForDepositAccounts = new
    DepositAccountPercentage();
44                  for (int i = 0; i < 3; i++) {
45                      Scanner in2 = new Scanner(System.in);
46                      String str = in2.nextLine();
47                      double first = Double.parseDouble(str.split("
    ")[0]);
48                      double second = Double.parseDouble(str.split("
     ")[1]);
49                      percentageOnBalanceForDepositAccounts.
    addParametersForDepositAccountBank(first, second);
50                  }
51
52                  System.out.println("Set
    percentageOnBalanceForDebitAccounts\nExample: double");
53                  double percentageOnBalanceForDebitAccounts = in.
    nextDouble();
54                  bank = centralBank.addBankToBase(
55                          name,
56                          limitForNotVerification,
57                          creditLimitForCreditAccounts,
58                          commissionUsingForCreditAccounts,
59                          percentageOnBalanceForDepositAccounts,
60                          percentageOnBalanceForDebitAccounts);
61                  System.out.println("Done");
62                  break;
63              }
64
65              case "2":
66                  System.out.println("1 - Start Create Client");
67                  System.out.println("2 - Start Create Verification
    Client");
68                  b = in.nextLine();
69                  switch (b) {
70                      case "1": {
71                          System.out.println("Set name\nExample:
    string");
72                          String name = in.nextLine();
73                          System.out.println("Set surname\nExample:
    string");
74                          String surname = in.nextLine();
75                          client = Client.Builder(name, surname).
    getClient();
76                          System.out.println("Done");
77                          break;
78                      }
79
80                      case "2": {
81                          System.out.println("Set name\nExample:
```

```
    string");
82                              String name = in.nextLine();
83                              System.out.println("Set surname\nExample:
    string");
84                              String surname = in.nextLine();
85                              System.out.println("Set address\nExample:
    string");
86                              String address = in.nextLine();
87                              System.out.println("Set passport\nExample:
     string");
88                              String passport = in.nextLine();
89                              client = Client.Builder(name, surname).
    addAddress(address).addPassport(passport).getClient();
90                              System.out.println("Done");
91                              break;
92                          }
93                      }
94
95                      break;
96                  case "3":
97                      System.out.println("1 - Start Create Debit");
98                      System.out.println("2 - Start Create Credit");
99                      System.out.println("3 - Start Create Deposit");
100                     b = in.nextLine();
101                     switch (b) {
102                         case "1":
103                             account1 = centralBank.
    regAccountClientInBank(bank, client, AccountOption.Debit, 10000);
104                             account2 = centralBank.
    regAccountClientInBank(bank, client, AccountOption.Debit, 10000);
105                             System.out.println("Done");
106                             break;
107                         case "2":
108                             account1 = centralBank.
    regAccountClientInBank(bank, client, AccountOption.Credit, 10000);
109                             account2 = centralBank.
    regAccountClientInBank(bank, client, AccountOption.Credit, 10000);
110                             System.out.println("Done");
111                             break;
112                         case "3":
113                             account1 = centralBank.
    regAccountClientInBank(bank, client, AccountOption.Deposit, 10000);
114                             account2 = centralBank.
    regAccountClientInBank(bank, client, AccountOption.Deposit, 10000);
115                             System.out.println("Done");
116                             break;
117                     }
118
119                     break;
120                 case "4":
```

```
121                     System.out.println("Set Amount Transaction\
    nExample: double");
122                     double amount = in.nextDouble();
123                     System.out.println("1 - Do Replenishment Money
    Transaction");
124                     System.out.println("2 - Do Withdrawal Money
    Transaction");
125                     System.out.println("3 - Do Transfer Money
    Transaction");
126                     b = in.nextLine();
127                     switch (b) {
128                         case "1":
129                             transaction = centralBank.
    replenishmentMoney(account1, amount);
130                             System.out.println("Done");
131                             break;
132                         case "2":
133                             transaction = centralBank.withdrawalMoney(
    account1, amount);
134                             System.out.println("Done");
135                             break;
136                         case "3":
137                             transaction = centralBank.transferMoney(
    account1, account2, amount);
138                             System.out.println("Done");
139                             break;
140                     }
141
142                     break;
143                 case "5":
144                     centralBank.cancelTransaction(transaction);
145                     System.out.println("Done");
146                     break;
147             }
148         }
149     }
150 }
```

Листинг 1.15: EventRegistrar.java

```
1  package tools;
2
3  import clientServices.Client;
4
5  import java.util.ArrayList;
6  import java.util.HashMap;
7
8  public class EventRegistrar {
9      HashMap<String, ArrayList<Client>> listeners = new HashMap<>();
10
11     public EventRegistrar(String... operations) {
12         for (String operation : operations) {
13             this.listeners.put(operation, new ArrayList<>());
14         }
15     }
16
17     public void subscribe(String eventType, Client listener) {
18         ArrayList<Client> users = listeners.get(eventType);
19         users.add(listener);
20     }
21
22     public void subscribeAll(Client listener) {
23         for (ArrayList<Client> clients : listeners.values())
24             clients.add(listener);
25     }
26
27     public void unsubscribe(String eventType, Client listener) {
28         ArrayList<Client> users = listeners.get(eventType);
29         users.remove(listener);
30     }
31
32     public void notify(String eventType) {
33         ArrayList<Client> users = listeners.get(eventType);
34         for (Client listener : users) {
35             Client.update(eventType);
36         }
37     }
38 }
```

Листинг 1.16: Pair.java

```
1  package tools;
2
3  public class Pair{
4      private double sum;
5      private double percentage;
6      public Pair(double sum, double percentage){
7          this.sum = sum;
8          this.percentage = percentage;
9      }
10     public double getSum(){ return sum; }
11     public double getPercentage(){ return percentage; }
12     public void setSum(double sum){ this.sum = sum; }
13     public void setPercentage(double percentage){ this.percentage =
       percentage; }
14 }
```

Листинг 1.17: Application.java

```
package com.ferbator;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Листинг 1.18: WebSecurityConfig.java

```
1  package com.ferbator.configuration;
2
3  import com.ferbator.services.WebSecurityService;
4  import org.springframework.beans.factory.annotation.Autowired;
5  import org.springframework.context.annotation.Bean;
6  import org.springframework.security.authentication.dao.
       DaoAuthenticationProvider;
7  import org.springframework.security.config.annotation.web.builders.
       HttpSecurity;
8  import org.springframework.security.config.annotation.web.
       configuration.EnableWebSecurity;
9  import org.springframework.security.config.annotation.web.
       configuration.WebSecurityConfigurerAdapter;
10 import org.springframework.security.crypto.bcrypt.
       BCryptPasswordEncoder;
11 import org.springframework.security.crypto.password.PasswordEncoder;
12 import org.springframework.stereotype.Component;
13
14 @EnableWebSecurity
15 @Component
16 public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
17
18     @Autowired
19     WebSecurityService service;
20
21     protected void configure(HttpSecurity http) throws Exception {
22         http.httpBasic()
23                 .and()
24                 .authorizeRequests()
25                 .antMatchers("/admin/**").hasRole("ADMIN")
26                 .antMatchers("/user/**").authenticated()
27                 .and()
28                 .logout().logoutSuccessUrl("/")
29                 .and()
30                 .csrf().disable()
31                 .formLogin();
32     }
33
34     @Bean
35     public DaoAuthenticationProvider daoAuthenticationProvider(){
36         DaoAuthenticationProvider authenticationProvider = new
       DaoAuthenticationProvider();
37         authenticationProvider.setPasswordEncoder(passwordEncoder());
38         authenticationProvider.setUserDetailsService(service);
39         return authenticationProvider;
40     }
41
42     private PasswordEncoder passwordEncoder(){
43         return new BCryptPasswordEncoder();
```

```
44          }
45  }
```

Листинг 1.19: AdminController.java

```
package com.ferbator.controller;

import com.ferbator.dao.dto.FriendshipCatDto;
import com.ferbator.dao.dto.OwnerDto;
import com.ferbator.dao.dto.OwnershipCatDto;
import com.ferbator.services.ShelterService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/admin")
public class AdminController {
    @Autowired
    ShelterService service;

    @GetMapping("")
    public String adminPage() {
        return "Shelter service for admin";
    }

    @PostMapping("/add-owner")
    public boolean addOwner(@RequestBody OwnerDto ownerDTO) {
        return service.addOwner(ownerDTO);
    }

    @DeleteMapping("/delete-owner/{id}")
    public boolean delOwnerById(@PathVariable("id") Long id) {
        return service.delOwner(id);
    }

    @GetMapping("/find-all-owner")
    public List<OwnerDto> findAllOwners() {
        return service.getListAllOwners();
    }

    @DeleteMapping("/break-ownership")
    public boolean breakOwnershipCat(@RequestBody OwnershipCatDto
    ownershipCat) {
        return service.breakOwnershipCat(ownershipCat.getOwnerId(),
    ownershipCat.getCatId());
    }

    @PostMapping("/make-ownership")
    public boolean makeOwnershipCat(@RequestBody OwnershipCatDto
    ownershipCat) {
        return service.makeOwnershipCat(ownershipCat);
    }
```

```
47
48      @DeleteMapping("/break-friendship")
49      public boolean breakFriendship(@RequestBody FriendshipCatDto
    friendshipCat) {
50          return service.breakFriendshipCat(friendshipCat.getFirstCatId
    (), friendshipCat.getSecondCatId());
51      }
52
53      @PostMapping("/make-friendship")
54      public boolean makeFriendship(@RequestBody FriendshipCatDto
    friendshipCat) {
55          return service.makeFriendshipCat(friendshipCat);
56      }
57  }
```

Листинг 1.20: HomeController.java

```
1  package com.ferbator.controller;
2
3  import org.springframework.web.bind.annotation.GetMapping;
4  import org.springframework.web.bind.annotation.RestController;
5
6  @RestController
7  public class HomeController {
8      @GetMapping("")
9      public String homePage() {
10         return "Shelter service";
11     }
12 }
```

Листинг 1.21: OwnerController.java

```
package com.ferbator.controller;

import com.ferbator.dao.dto.CatDto;
import com.ferbator.dao.dto.FriendshipCatDto;
import com.ferbator.dao.dto.OwnershipCatDto;
import com.ferbator.dao.enums.Colors;
import com.ferbator.services.ShelterService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.server.ResponseStatusException;

import java.util.List;

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

    @GetMapping("/find-all-cats")
    public List<CatDto> findAllCats() {
        return service.getListAllCats();
    }

    @GetMapping("/find-all-one-color-cats/{color}")
    public List<CatDto> findAllOneColorCats(@PathVariable("color")
   String color) {
        try {
            return service.getListAllOneColorCats(Colors.valueOf(color
   ));
        } catch (IllegalArgumentException e) {
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST);
        }
    }

    @PostMapping("/add-cat")
    public boolean addCat(@RequestBody CatDto catDTO) {
        return service.addCat(catDTO);
    }

    @DeleteMapping("/delete-cat/{id}")
    public boolean delCatById(@PathVariable("id") Long id) {
        return service.delCat(id);
    }

    @DeleteMapping("/break-ownership")
    public boolean breakOwnershipCat(@RequestBody OwnershipCatDto
   ownershipCat) {
```

```
47          return service.breakOwnershipCat(ownershipCat.getOwnerId(),
    ownershipCat.getCatId());
48      }
49
50      @PostMapping("/make-ownership")
51      public boolean makeOwnershipCat(@RequestBody OwnershipCatDto
    ownershipCat) {
52          return service.makeOwnershipCat(ownershipCat);
53      }
54
55      @DeleteMapping("/break-friendship")
56      public boolean breakFriendship(@RequestBody FriendshipCatDto
    friendshipCat) {
57          return service.breakFriendshipCat(friendshipCat.getFirstCatId
    (), friendshipCat.getSecondCatId());
58      }
59
60      @PostMapping("/make-friendship")
61      public boolean makeFriendship(@RequestBody FriendshipCatDto
    friendshipCat) {
62          return service.makeFriendshipCat(friendshipCat);
63      }
64  }
```

Листинг 1.22: RegistrationController.java

```java
package com.ferbator.controller;

import com.ferbator.dao.dto.OwnerDto;
import com.ferbator.services.ShelterService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController("registrationController")
@RequestMapping("/registration")
public class RegistrationController {

    @Autowired
    ShelterService service;


    @PostMapping()
    public String registration(@RequestBody OwnerDto owner){
        try {
            service.addOwner(owner);
            return "OK";
        } catch (Exception e) {
            return "ERROR: " + e.getMessage();
        }
    }
}
```

Листинг 1.23: CatRepository.java

```
1  package com.ferbator.dao.daoImpl;
2
3  import com.ferbator.dao.entities.Cat;
4  import com.ferbator.dao.enums.Colors;
5  import org.springframework.data.jpa.repository.JpaRepository;
6  import org.springframework.stereotype.Repository;
7
8  import java.util.List;
9
10 @Repository
11 public interface CatRepository extends JpaRepository<Cat, Long> {
12     List<Cat> findAllByColor(Colors color);
13 }
```

Листинг 1.24: FriendshipCatRepository.java

```
1  package com.ferbator.dao.daoImpl;
2
3  import com.ferbator.dao.entities.FriendshipCat;
4  import org.springframework.data.jpa.repository.JpaRepository;
5  import org.springframework.stereotype.Repository;
6
7  @Repository
8  public interface FriendshipCatRepository extends JpaRepository<
       FriendshipCat, Long> {
9      void deleteAllByFirstCatIdOrSecondCatId(Long firstCatId, Long
       secondCatId);
10 }
```

Листинг 1.25: OwnerRepository.java

```
1  package com.ferbator.dao.daoImpl;
2
3  import com.ferbator.dao.entities.Owner;
4  import org.springframework.data.jpa.repository.JpaRepository;
5  import org.springframework.stereotype.Repository;
6
7  @Repository
8  public interface OwnerRepository extends JpaRepository<Owner, Long> {
9      Owner findByLogin(String login);
10 }
```

Листинг 1.26: OwnershipCatRepository.java

```
1  package com.ferbator.dao.daoImpl;
2
3  import com.ferbator.dao.entities.OwnershipCat;
4  import org.springframework.data.jpa.repository.JpaRepository;
5  import org.springframework.stereotype.Repository;
6
7
8  @Repository
9  public interface OwnershipCatRepository extends JpaRepository<
       OwnershipCat, Long> {
10     void deleteAllByCatId(Long catId);
11     void deleteAllByOwnerId(Long ownerId);
12 }
```

Листинг 1.27: CatDto.java

```
package com.ferbator.dao.dto;

import com.ferbator.dao.entities.Cat;
import com.ferbator.dao.enums.Colors;

import java.sql.Timestamp;
import java.util.Objects;

public class CatDto {
    private Long id;
    private Colors color;
    private String name;
    private String breed;
    private Timestamp birthday;

    public CatDto() {
    }

    public CatDto(Cat cat) {
        this.id = cat.getId();
        this.name = cat.getName();
        this.color = cat.getColor();
        this.breed = cat.getBreed();
        this.birthday = cat.getBirthday();
    }

    public Long getId() {
        return id;
    }

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

    public Colors getColor() {
        return color;
    }

    public void setColor(Colors color) {
        this.color = color;
    }

    public String getName() {
        return name;
    }

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

```
50
51      public String getBreed() {
52          return breed;
53      }
54
55      public void setBreed(String breed) {
56          this.breed = breed;
57      }
58
59      public Timestamp getBirthday() {
60          return birthday;
61      }
62
63      public void setBirthday(Timestamp birthday) {
64          this.birthday = birthday;
65      }
66
67      @Override
68      public boolean equals(Object o) {
69          if (this == o) return true;
70          if (o == null || getClass() != o.getClass()) return false;
71          CatDto catDTO = (CatDto) o;
72          return Objects.equals(id, catDTO.id)
73                  && color == catDTO.color
74                  && Objects.equals(name, catDTO.name)
75                  && Objects.equals(breed, catDTO.breed)
76                  && Objects.equals(birthday, catDTO.birthday);
77      }
78
79      @Override
80      public int hashCode() {
81          return Objects.hash(id, color, name, breed, birthday);
82      }
83  }
```

Листинг 1.28: FriendshipCatDto.java

```java
package com.ferbator.dao.dto;

import com.ferbator.dao.entities.FriendshipCat;

import java.util.Objects;

public class FriendshipCatDto {
    private Long id;
    private Long firstCatId;
    private Long secondCatId;

    public FriendshipCatDto() {
    }

    public FriendshipCatDto(FriendshipCat friendshipCat) {
        this.id = friendshipCat.getId();
        this.firstCatId = friendshipCat.getFirstCatId();
        this.secondCatId = friendshipCat.getSecondCatId();
    }

    public Long getId() {
        return id;
    }

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

    public Long getFirstCatId() {
        return firstCatId;
    }

    public void setFirstCatId(Long firstCatId) {
        this.firstCatId = firstCatId;
    }

    public Long getSecondCatId() {
        return secondCatId;
    }

    public void setSecondCatId(Long secondCatId) {
        this.secondCatId = secondCatId;
    }

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

```
50          return Objects.equals(id, that.id)
51                  && Objects.equals(firstCatId, that.firstCatId)
52                  && Objects.equals(secondCatId, that.secondCatId);
53      }
54
55      @Override
56      public int hashCode() {
57          return Objects.hash(id, firstCatId, secondCatId);
58      }
59  }
```

Листинг 1.29: OwnerDto.java

```
1  package com.ferbator.dao.dto;
2
3  import com.ferbator.dao.entities.Owner;
4
5  import javax.persistence.Basic;
6  import javax.persistence.Column;
7  import java.sql.Timestamp;
8  import java.util.Objects;
9
10 public class OwnerDto {
11     private Long id;
12     private String name;
13     private Timestamp birthday;
14     private String login;
15     private String password;
16     private String role;
17
18     public OwnerDto(Owner own) {
19         this.id = own.getId();
20         this.name = own.getName();
21         this.birthday = own.getBirthday();
22         this.login = own.getLogin();
23         this.password = own.getPassword();
24         this.role = own.getRole();
25     }
26
27     public OwnerDto() {
28     }
29
30     public Long getId() {
31         return id;
32     }
33
34     public void setId(Long id) {
35         this.id = id;
36     }
37
38     public String getName() {
39         return name;
40     }
41
42     public void setName(String login) {
43         this.login = login;
44     }
45
46     public String getLogin() {
47         return login;
48     }
49
```

```
50      public void setLogin(String name) {
51          this.name = name;
52      }
53
54      public String getPassword() {
55          return password;
56      }
57
58      public void setPassword(String password) {
59          this.password = password;
60      }
61
62      public String getRole() {
63          return role;
64      }
65
66      public void setRole(String role) {
67          this.role = role;
68      }
69
70      public Timestamp getBirthday() {
71          return birthday;
72      }
73
74      public void setBirthday(Timestamp birthday) {
75          this.birthday = birthday;
76      }
77
78      @Override
79      public boolean equals(Object o) {
80          if (this == o) return true;
81          if (o == null || getClass() != o.getClass()) return false;
82          OwnerDto ownerDto = (OwnerDto) o;
83          return Objects.equals(id, ownerDto.id) && Objects.equals(name,
       ownerDto.name) && Objects.equals(birthday, ownerDto.birthday) &&
      Objects.equals(login, ownerDto.login) && Objects.equals(password,
      ownerDto.password) && Objects.equals(role, ownerDto.role);
84      }
85
86      @Override
87      public int hashCode() {
88          return Objects.hash(id, name, birthday, login, password, role)
      ;
89      }
90  }
```

Листинг 1.30: OwnershipCatDto.java

```
1  package com.ferbator.dao.dto;
2
3  import com.ferbator.dao.entities.OwnershipCat;
4
5  import java.util.Objects;
6
7  public class OwnershipCatDto {
8      private Long id;
9      private Long ownerId;
10     private Long catId;
11
12     public OwnershipCatDto() {
13     }
14
15     public OwnershipCatDto(OwnershipCat ownershipCat) {
16         this.ownerId = ownershipCat.getOwnerId();
17         this.catId = ownershipCat.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
44     @Override
45     public boolean equals(Object o) {
46         if (this == o) return true;
47         if (o == null || getClass() != o.getClass()) return false;
48         OwnershipCatDto that = (OwnershipCatDto) o;
49         return Objects.equals(id, that.id)
```

```
50                  && Objects.equals(ownerId, that.ownerId)
51                  && Objects.equals(catId, that.catId);
52      }
53
54      @Override
55      public int hashCode() {
56          return Objects.hash(id, ownerId, catId);
57      }
58  }
```

Листинг 1.31: Cat.java

```
1  package com.ferbator.dao.entities;
2
3  import com.ferbator.dao.enums.Colors;
4  import com.ferbator.dao.dto.CatDto;
5
6  import javax.persistence.*;
7  import java.sql.Timestamp;
8  import java.util.Objects;
9
10 @Entity
11 @Table(name = "cats")
12 public class Cat {
13     @Basic
14     @Column(name = "name", length = -1)
15     private String name;
16     @Basic
17     @Column(name = "birthday")
18     private Timestamp birthday;
19     @Basic
20     @Column(name = "breed", length = -1)
21     private String breed;
22     @GeneratedValue(strategy = GenerationType.IDENTITY)
23     @Id
24     @Column(name = "id", nullable = false)
25     private Long id;
26     @Basic
27     @Column(name = "color", length = -1)
28     @Enumerated(EnumType.STRING)
29     private Colors color;
30
31     public Cat(CatDto cat) {
32         this.name = cat.getName();
33         this.color = cat.getColor();
34         this.breed = cat.getBreed();
35         this.birthday = cat.getBirthday();
36     }
37
38     public Cat() {
39     }
40
41     public String getName() {
42         return name;
43     }
44
45     public void setName(String name) {
46         this.name = name;
47     }
48
49     public Timestamp getBirthday() {
```

```
50          return birthday;
51      }
52
53      public void setBirthday(Timestamp birthday) {
54          this.birthday = birthday;
55      }
56
57      public String getBreed() {
58          return breed;
59      }
60
61      public void setBreed(String breed) {
62          this.breed = breed;
63      }
64
65      public Long getId() {
66          return id;
67      }
68
69      public void setId(Long id) {
70          this.id = id;
71      }
72
73      public Colors getColor() {
74          return color;
75      }
76
77      public void setColor(Colors color) {
78          this.color = color;
79      }
80
81      @Override
82      public boolean equals(Object o) {
83          if (this == o) return true;
84          if (o == null || getClass() != o.getClass()) return false;
85          Cat cat = (Cat) o;
86          return Objects.equals(id, cat.id)
87                  && Objects.equals(name, cat.name)
88                  && Objects.equals(birthday, cat.birthday)
89                  && Objects.equals(breed, cat.breed)
90                  && Objects.equals(color, cat.color);
91      }
92
93      @Override
94      public int hashCode() {
95          return Objects.hash(name, birthday, breed, id, color);
96      }
97  }
```

Листинг 1.32: FriendshipCat.java

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

```
50      public void setSecondCatId(Long secondCatId) {
51          this.secondCatId = secondCatId;
52      }
53
54      @Override
55      public boolean equals(Object o) {
56          if (this == o) return true;
57          if (o == null || getClass() != o.getClass()) return false;
58          FriendshipCat that = (FriendshipCat) o;
59          return Objects.equals(id, that.id)
60                  && Objects.equals(firstCatId, that.firstCatId)
61                  && Objects.equals(secondCatId, that.secondCatId);
62      }
63
64      @Override
65      public int hashCode() {
66          return Objects.hash(id, firstCatId, secondCatId);
67      }
68  }
```

Листинг 1.33: Owner.java

```
1  package com.ferbator.dao.entities;
2
3  import com.ferbator.dao.dto.OwnerDto;
4
5  import javax.persistence.*;
6  import java.sql.Timestamp;
7  import java.util.Objects;
8
9  @Entity
10 @Table(name = "owners")
11 public class Owner {
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     @Id
14     @Column(name = "id", nullable = false)
15     private Long id;
16     @Basic
17     @Column(name = "name", length = -1)
18     private String name;
19     @Basic
20     @Column(name = "birthday")
21     private Timestamp birthday;
22     @Basic
23     @Column(name = "login", length = -1)
24     private String login;
25     @Basic
26     @Column(name = "password", length = -1)
27     private String password;
28     @Basic
29     @Column(name = "role", length = -1)
30     private String role;
31
32     public Owner(OwnerDto own) {
33         this.name = own.getName();
34         this.birthday = own.getBirthday();
35         this.login = own.getLogin();
36         this.password = own.getPassword();
37         this.role = own.getRole();
38     }
39
40     public Owner() {
41     }
42
43     public Long getId() {
44         return id;
45     }
46
47     public void setId(Long id) {
48         this.id = id;
49     }
```

```
50
51      public String getName() {
52          return name;
53      }
54
55      public void setName(String name) {
56          this.name = name;
57      }
58
59      public Timestamp getBirthday() {
60          return birthday;
61      }
62
63      public void setBirthday(Timestamp birthday) {
64          this.birthday = birthday;
65      }
66
67      public String getLogin() {
68          return login;
69      }
70
71      public void setLogin(String name) {
72          this.name = name;
73      }
74
75      public String getPassword() {
76          return password;
77      }
78
79      public void setPassword(String password) {
80          this.password = password;
81      }
82
83      public String getRole() {
84          return role;
85      }
86
87      public void setRole(String role) {
88          this.role = role;
89      }
90
91      @Override
92      public boolean equals(Object o) {
93          if (this == o) return true;
94          if (o == null || getClass() != o.getClass()) return false;
95          Owner owner = (Owner) o;
96          return Objects.equals(id, owner.id) && Objects.equals(name,
    owner.name) && Objects.equals(birthday, owner.birthday) && Objects.
    equals(login, owner.login) && Objects.equals(password, owner.
    password) && Objects.equals(role, owner.role);
```

```
97      }
98
99      @Override
100     public int hashCode() {
101         return Objects.hash(id, name, birthday, login, password, role)
        ;
102     }
103 }
```

Листинг 1.34: OwnershipCat.java

```
1  package com.ferbator.dao.entities;
2
3  import com.ferbator.dao.dto.OwnershipCatDto;
4
5  import javax.persistence.*;
6  import java.util.Objects;
7
8  @Entity
9  @Table(name = "ownershipcats")
10 public class OwnershipCat {
11     @GeneratedValue(strategy = GenerationType.IDENTITY)
12     @Id
13     @Column(name = "id", nullable = false)
14     private Long id;
15     @Basic
16     @Column(name = "owner_id", nullable = false)
17     private Long ownerId;
18     @Basic
19     @Column(name = "cat_id", nullable = false)
20     private Long catId;
21
22     public OwnershipCat(OwnershipCatDto ownershipCat) {
23         this.ownerId = ownershipCat.getOwnerId();
24         this.catId = ownershipCat.getCatId();
25     }
26
27     public OwnershipCat() {
28     }
29
30     public Long getId() {
31         return id;
32     }
33
34     public void setId(Long id) {
35         this.id = id;
36     }
37
38     public Long getOwnerId() {
39         return ownerId;
40     }
41
42     public void setOwnerId(Long ownerId) {
43         this.ownerId = ownerId;
44     }
45
46     public Long getCatId() {
47         return catId;
48     }
49
```

```
50      public void setCatId(Long catId) {
51          this.catId = catId;
52      }
53
54      @Override
55      public boolean equals(Object o) {
56          if (this == o) return true;
57          if (o == null || getClass() != o.getClass()) return false;
58          OwnershipCat that = (OwnershipCat) o;
59          return Objects.equals(id, that.id)
60                  && Objects.equals(ownerId, that.ownerId)
61                  && Objects.equals(catId, that.catId);
62      }
63
64      @Override
65      public int hashCode() {
66          return Objects.hash(id, ownerId, catId);
67      }
68  }
```

Листинг 1.35: Colors.java

```
1 package com.ferbator.dao.enums;
2
3 public enum Colors {
4     Black,
5     Red,
6     Orange,
7     Blue
8 }
```

Листинг 1.36: DAOException.java

```
1  package com.ferbator.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 1.37: ShelterService.java

```
package com.ferbator.services;

import com.ferbator.dao.daoImpl.CatRepository;
import com.ferbator.dao.daoImpl.FriendshipCatRepository;
import com.ferbator.dao.daoImpl.OwnerRepository;
import com.ferbator.dao.daoImpl.OwnershipCatRepository;
import com.ferbator.dao.dto.FriendshipCatDto;
import com.ferbator.dao.dto.OwnerDto;
import com.ferbator.dao.dto.OwnershipCatDto;
import com.ferbator.dao.entities.Cat;
import com.ferbator.dao.dto.CatDto;
import com.ferbator.dao.entities.FriendshipCat;
import com.ferbator.dao.entities.Owner;
import com.ferbator.dao.entities.OwnershipCat;
import com.ferbator.dao.enums.Colors;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.
    SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.crypto.bcrypt.
    BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import javax.transaction.Transactional;
import java.util.Collection;
import java.util.List;
import java.util.stream.Collectors;

@Service
public class ShelterService {
    CatRepository catRepository;
    OwnerRepository ownerRepository;
    FriendshipCatRepository friendshipCatRepository;
    OwnershipCatRepository ownershipCatRepository;

    @Autowired
    public ShelterService(CatRepository catRepository, OwnerRepository
    ownerRepository, FriendshipCatRepository friendshipCatRepository,
    OwnershipCatRepository ownershipCatRepository) {
        this.catRepository = catRepository;
        this.ownerRepository = ownerRepository;
        this.friendshipCatRepository = friendshipCatRepository;
        this.ownershipCatRepository = ownershipCatRepository;
    }

    public List<CatDto> getListAllCats() {
        return catRepository.findAll().stream().map(CatDto::new).
```

```
    toList();
46   }
47
48   public List<OwnerDto> getListAllOwners() {
49       return ownerRepository.findAll().stream().map(OwnerDto::new).
    toList();
50   }
51
52   public List<CatDto> getListAllOneColorCats(Colors colors) {
53       return catRepository.findAllByColor(colors).stream().map(
    CatDto::new).toList();
54   }
55
56   public boolean addCat(CatDto cat) {
57       catRepository.save(new Cat(cat));
58       return true;
59   }
60
61   public boolean delCat(Long id) {
62       ownershipCatRepository.deleteAllByCatId(id);
63       friendshipCatRepository.deleteAllByFirstCatIdOrSecondCatId(id,
    id);
64       catRepository.deleteById(id);
65       return true;
66   }
67
68   public boolean addOwner(OwnerDto owner) {
69       ownerRepository.save(new Owner(owner));
70       return true;
71   }
72
73   public OwnerDto findOwnerByLogin(String login) {
74       return new OwnerDto(ownerRepository.findByLogin(login));
75   }
76
77   @Transactional
78   public boolean delOwner(Long id) {
79       ownershipCatRepository.deleteAllByOwnerId(id);
80       ownerRepository.deleteById(id);
81       return true;
82   }
83
84   public boolean breakOwnershipCat(Long ownerId, Long catId){
85       OwnershipCatDto tmpOwnershipCatDto = new OwnershipCatDto();
86       tmpOwnershipCatDto.setCatId(catId);
87       tmpOwnershipCatDto.setOwnerId(ownerId);
88       ownershipCatRepository.save(new OwnershipCat(
    tmpOwnershipCatDto));
89       return true;
90   }
```

```
91
92      public boolean makeOwnershipCat(OwnershipCatDto ownershipCat){
93          ownershipCatRepository.save(new OwnershipCat(ownershipCat));
94          return true;
95      }
96
97      public boolean breakFriendshipCat(Long firstCatId, Long
    secondCatId){
98          FriendshipCatDto tmpFriendshipCatDto = new FriendshipCatDto();
99          tmpFriendshipCatDto.setFirstCatId(firstCatId);
100         tmpFriendshipCatDto.setSecondCatId(secondCatId);
101         friendshipCatRepository.save(new FriendshipCat(
    tmpFriendshipCatDto));
102         return true;
103     }
104
105     public boolean makeFriendshipCat(FriendshipCatDto friendshipCat){
106         friendshipCatRepository.save(new FriendshipCat(friendshipCat))
    ;
107         return true;
108     }
109 }
```

Листинг 1.38: WebSecurityService.java

```
1  package com.ferbator.services;
2
3  import com.ferbator.dao.dto.OwnerDto;
4  import org.springframework.beans.factory.annotation.Autowired;
5  import org.springframework.security.core.GrantedAuthority;
6  import org.springframework.security.core.authority.
       SimpleGrantedAuthority;
7  import org.springframework.security.core.userdetails.User;
8  import org.springframework.security.core.userdetails.UserDetails;
9  import org.springframework.security.core.userdetails.
       UserDetailsService;
10 import org.springframework.security.core.userdetails.
       UsernameNotFoundException;
11 import org.springframework.stereotype.Service;
12 import org.springframework.transaction.annotation.Transactional;
13
14
15 import java.util.Collection;
16 import java.util.List;
17 import java.util.stream.Collectors;
18
19 @Service("webSecurityService")
20 public class WebSecurityService implements UserDetailsService {
21
22     @Autowired
23     private ShelterService service;
24
25     public Collection<? extends GrantedAuthority>
       mapRolesToAuthorities(Collection<String> roles){
26         return roles.stream().map(SimpleGrantedAuthority::new).collect
       (Collectors.toList());
27     }
28
29     public User mapOwnerDtoToUserDetails(OwnerDto owner){
30         return new User(owner.getLogin(), owner.getPassword(),
       mapRolesToAuthorities(List.of(owner.getRole())));
31     }
32
33     @Override
34     @Transactional
35     public UserDetails loadUserByUsername(String login) throws
       UsernameNotFoundException {
36         OwnerDto owner = service.findOwnerByLogin(login);
37         if(owner == null){
38             throw new UsernameNotFoundException("Owner doesn't exist")
       ;
39         }
40         return mapOwnerDtoToUserDetails(owner);
41     }
```

```
42 }
```

Листинг 1.39: ShelterServiceException.java

```
package com.ferbator.services.tools;

public class ShelterServiceException extends Exception {
    public ShelterServiceException() {
        super();
    }

    public ShelterServiceException(String message) {
        super(message);
    }

    public ShelterServiceException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

Листинг 1.40: OwnerControllerTest.java

```java
package com.ferbator.controller;

import com.ferbator.dao.daoImpl.CatRepository;
import com.ferbator.dao.daoImpl.FriendshipCatRepository;
import com.ferbator.dao.daoImpl.OwnerRepository;
import com.ferbator.dao.daoImpl.OwnershipCatRepository;
import com.ferbator.dao.dto.OwnerDto;
import com.ferbator.dao.entities.Owner;
import com.ferbator.services.ShelterService;
import com.ferbator.services.WebSecurityService;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.
    WebMvcTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.security.core.authority.
    SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.
    MockMvcRequestBuilders;
import org.springframework.security.test.web.servlet.request.
    SecurityMockMvcRequestPostProcessors;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

import static org.springframework.test.web.servlet.result.
    MockMvcResultMatchers.status;

@WebMvcTest(OwnerController.class)
public class OwnerControllerTest {
    @MockBean
    ShelterService service;

    @MockBean
    OwnerRepository ownerRepository;

    @MockBean
    CatRepository catRepository;

    @MockBean
    FriendshipCatRepository friendshipCatRepository;

    @MockBean
    OwnershipCatRepository ownershipCatRepository;
```

```
45
46      @MockBean
47      WebSecurityService webSecurityService;
48
49      @Autowired
50      private MockMvc mockMvc;
51
52      private static Owner owner;
53      private static OwnerDto ownerDto;
54
55      @BeforeEach
56      public void setUp() {
57          owner = new Owner();
58          owner.setLogin("Denis");
59          owner.setPassword("$2a$12$QslpGcxXWuH3TS9dUirDB.
    NtB6IFiqbvq89zJfRCdWILHsuQF9Uz6");
60          owner.setRole("ROLE_USER");
61          ownerDto = new OwnerDto(owner);
62
63          List<SimpleGrantedAuthority> tmpList = Stream.of(ownerDto.
    getRole()).map(SimpleGrantedAuthority::new).collect(Collectors.
    toList());
64
65          Mockito.when(webSecurityService.mapOwnerDtoToUserDetails(
    ownerDto)).thenReturn(new User(ownerDto.getLogin(), ownerDto.
    getPassword(), tmpList));
66
67          Mockito.when(webSecurityService.loadUserByUsername("Denis")).
    thenReturn(webSecurityService.mapOwnerDtoToUserDetails(ownerDto));
68          Mockito.when(service.findOwnerByLogin("Denis")).thenReturn(
    ownerDto);
69          Mockito.when(ownerRepository.findByLogin("Denis")).thenReturn(
    owner);
70      }
71
72      @Test
73      public void shouldAllowPageWhenUserIsAdmin() throws Exception {
74          String url = "http://localhost:8080/admin";
75          mockMvc.perform(MockMvcRequestBuilders
76                          .get(url)
77                          .with(SecurityMockMvcRequestPostProcessors.
    user("Denis")
78                                  .password("0000")
79                                  .roles("USER")))
80                  .andExpect(status().isForbidden());
81      }
82  }
```