Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Лабороторная работа №2

Выполнил студент:

Карепин Денис Дмитриевич
группа: M32071

Проверил:

Чикишев Константин Максимович

Санкт-Петербург,
2022 г.

1.1. Текст задания

2 лабораторная
Нужно написать сервис по учету котиков и их владельцев.
Существующая информация о котиках:
- Имя
- Дата рождения
- Порода
- Цвет (один из заранее заданных вариантов)
- Хозяин
- Список котиков, с которыми дружит этот котик (из представленных в базе)
Существующая информация о хозяевах:
- Имя
- Дата рождения
- Список котиков
Сервис должен реализовывать архитектуру controller-service-dao.
Вся информация хранится в БД PostgreSQL. Для связи с БД должен использоваться Hibernate.
Проект должен собираться с помощью Maven или Gradle (на выбор студента). Слой доступа к данным и сервисный слой должны являться двумя разными модулями Maven/Gradle. При этом проект должен полностью собираться одной командой.
При тестировании рекомендуется использовать Mockito, чтобы избежать подключения к реальным базам данных. Фреймворк для тестирования рекомендуется Junit 5.
В данной лабораторной нельзя использовать Spring или подобные ему фреймворки.

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

```
1  package controller;
2
3  import dao.daoImpl.CatsDAO;
4  import dao.daoImpl.FriendshipCatsDAO;
5  import dao.daoImpl.OwnersDAO;
6  import dao.daoImpl.OwnershipCatsDAO;
7  import dao.daoInterface.DAO;
8  import dao.entities.Cat;
9  import dao.entities.FriendshipCat;
10 import dao.entities.Owners;
11 import dao.entities.OwnershipCat;
12 import dao.enums.Colors;
13 import services.ShelterService;
14 import services.tools.ShelterServiceException;
15
16 import java.util.Objects;
17 import java.util.Scanner;
18
19 public class ConsoleInterface {
20     private ShelterService service;
21
22     public ConsoleInterface() {
23         DAO<Owners> daoOwn = new OwnersDAO();
24         DAO<Cat> daoCat = new CatsDAO();
25         DAO<OwnershipCat> daoOwnShip = new OwnershipCatsDAO();
26         DAO<FriendshipCat> daoFriendShip = new FriendshipCatsDAO();
27         this.service = new ShelterService(daoOwn, daoCat, daoOwnShip,
       daoFriendShip);
28     }
29
30     public void input() throws ShelterServiceException {
31         String str = null;
32         while (!Objects.equals(str, "Q")) {
33             preamble();
34             Scanner in = new Scanner(System.in);
35             str = in.nextLine();
36             switch (str) {
37                 case "1" -> {
38                     createOwner();
39                 }
40                 case "2" ->{
41                     createCat();
42                 }
43                 case "3" ->{
44                     deleteOwner();
45                 }
46                 case "4" ->{
47                     deleteCat();
48                 }
```

```
49                  case "5" ->{
50                      startOwnership();
51                  }
52                  case "6" ->{
53                      cancelOwnership();
54                  }
55                  case "7" ->{
56                      startFriendship();
57                  }
58                  case "8" ->{
59                      cancelFriendship();
60                  }
61              }
62          }
63      }
64
65      private void preamble() {
66          System.out.println("1 - Start Create Owner");
67          System.out.println("2 - Start Create Cat");
68          System.out.println("3 - Delete Owner by id");
69          System.out.println("4 - Delete Cat by id");
70          System.out.println("5 - Start Ownership by id of Owner and id
        Cat");
71          System.out.println("6 - Cancel Ownership by id of Owner and id
         Cat");
72          System.out.println("7 - Start Friendship by id of Cats");
73          System.out.println("8 - Cancel Friendship by id of Cats");
74          System.out.println("Q - Exit Program");
75      }
76
77      private void createOwner() throws ShelterServiceException {
78          Scanner in = new Scanner(System.in);
79          System.out.println("Start create");
80          System.out.println("Set name");
81          String name = in.nextLine();
82          System.out.println("Set birthday example:2002-01-12");
83          String birth = in.nextLine();
84          service.addOwnerToBase(name, birth + " 00:00:00");
85      }
86
87      private void createCat() throws ShelterServiceException {
88          Scanner in = new Scanner(System.in);
89          System.out.println("Start create");
90          System.out.println("Set name");
91          String name = in.nextLine();
92          System.out.println("Colors{\n 1 - Black,\n" +
93                  "2 - Red,\n" +
94                  "3 - Orange,\n" +
95                  "4 - Blue}");
96          Colors tmpColor = null;
```

```
97          switch (in.nextLine()) {
98              case "1" -> {
99                  tmpColor = Colors.Black;
100             }
101             case "2" -> {
102                 tmpColor = Colors.Red;
103             }
104             case "3" -> {
105                 tmpColor = Colors.Orange;
106             }
107             case "4" -> {
108                 tmpColor = Colors.Blue;
109             }
110         }
111         System.out.println("Set birthday example:2002-01-12");
112         String birth = in.nextLine();
113         System.out.println("Set breed");
114         String breed = in.nextLine();
115         service.addCatToBase(name, tmpColor, breed, birth + " 00:00:00
    ");
116     }
117
118     private void deleteOwner() throws ShelterServiceException{
119         Scanner in = new Scanner(System.in);
120         System.out.println("Start delete");
121         System.out.println("Set id");
122         long id = in.nextLong();
123         service.delOwnerFromBase(id);
124     }
125
126     private void deleteCat() throws ShelterServiceException{
127         Scanner in = new Scanner(System.in);
128         System.out.println("Start delete");
129         System.out.println("Set id");
130         long id = in.nextLong();
131         service.delCatFromBase(id);
132     }
133
134     private void startOwnership() throws ShelterServiceException{
135         Scanner in = new Scanner(System.in);
136         System.out.println("Start ownership");
137         System.out.println("Set id owner");
138         long id1 = in.nextLong();
139         System.out.println("Set id cat");
140         long id2 = in.nextLong();
141         service.PЎstartatOwnership(id1, id2);
142     }
143
144     private void cancelOwnership() throws ShelterServiceException{
145         Scanner in = new Scanner(System.in);
```

```
146             System.out.println("Cancel ownership");
147             System.out.println("Set id owner");
148             long id1 = in.nextLong();
149             System.out.println("Set id cat");
150             long id2 = in.nextLong();
151             service.cancelCatOwnership(id1, id2);
152         }
153 
154         private void startFriendship() throws ShelterServiceException{
155             Scanner in = new Scanner(System.in);
156             System.out.println("Start friendship");
157             System.out.println("Set id first cat");
158             long id1 = in.nextLong();
159             System.out.println("Set id second cat");
160             long id2 = in.nextLong();
161             service.PŬstartatFriendship(id1, id2);
162         }
163 
164         private void cancelFriendship() throws ShelterServiceException{
165             Scanner in = new Scanner(System.in);
166             System.out.println("Cancel friendship");
167             System.out.println("Set id first cat");
168             long id1 = in.nextLong();
169             System.out.println("Set id second cat");
170             long id2 = in.nextLong();
171             service.cancelCatFriendship(id1, id2);
172         }
173 }
```

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: Main.java

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

Листинг 1.18: CatsDAO.java

```
1  package dao.daoImpl;
2
3  import dao.daoInterface.DAO;
4  import dao.entities.Cat;
5  import dao.tools.DAOException;
6  import dao.tools.HibernateUtil;
7  import org.hibernate.HibernateException;
8  import org.hibernate.Session;
9
10 import java.util.List;
11
12 public class CatsDAO implements DAO<Cat> {
13     @Override
14     public List<Cat> findAll() throws DAOException {
15         try {
16             List<Cat> objects;
17             Session session = HibernateUtil.getSessionFactory().
       openSession();
18             session.getTransaction().begin();
19             objects = session.createQuery("select e from Cat e order
       by e.id", Cat.class)
20                     .getResultList();
21             session.getTransaction().commit();
22             session.close();
23
24             return objects;
25         } catch (HibernateException e) {
26             throw new DAOException(e.getMessage(), e);
27         }
28     }
29
30     @Override
31     public boolean refract(Cat object) throws DAOException {
32         try {
33             Session session = HibernateUtil.getSessionFactory().
       openSession();
34             session.getTransaction().begin();
35             session.update(object);
36             session.getTransaction().commit();
37             session.close();
38
39             return true;
40         } catch (HibernateException e) {
41             throw new DAOException(e.getMessage(), e);
42         }
43     }
44
45     @Override
46     public boolean add(Cat object) throws DAOException {
```

```
47          try {
48              Session session = HibernateUtil.getSessionFactory().
    openSession();
49              session.getTransaction().begin();
50              session.save(object);
51              session.getTransaction().commit();
52              session.close();
53
54              return true;
55          } catch (HibernateException e) {
56              throw new DAOException(e.getMessage(), e);
57          }
58      }
59
60      @Override
61      public boolean del(Cat object) throws DAOException {
62          try {
63              Session session = HibernateUtil.getSessionFactory().
    openSession();
64              session.getTransaction().begin();
65              session.delete(object);
66              session.getTransaction().commit();
67              session.close();
68
69              return true;
70          } catch (HibernateException e) {
71              throw new DAOException(e.getMessage(), e);
72          }
73      }
74
75      @Override
76      public Cat getById(long id) throws DAOException {
77          try {
78              Session session = HibernateUtil.getSessionFactory().
    openSession();
79              Cat item = session.byId(Cat.class).load(id);
80              session.close();
81
82              return item;
83          } catch (HibernateException e) {
84              throw new DAOException(e.getMessage(), e);
85          }
86      }
87  }
```

Листинг 1.19: FriendshipCatsDAO.java

```
1  package dao.daoImpl;
2
3  import dao.daoInterface.DAO;
4  import dao.entities.FriendshipCat;
5  import dao.tools.DAOException;
6  import dao.tools.HibernateUtil;
7  import org.hibernate.HibernateException;
8  import org.hibernate.Session;
9
10 import java.util.List;
11
12 public class FriendshipCatsDAO implements DAO<FriendshipCat> {
13     @Override
14     public List<FriendshipCat> findAll() throws DAOException {
15         try {
16             List<FriendshipCat> objects;
17             Session session = HibernateUtil.getSessionFactory().
    openSession();
18             session.getTransaction().begin();
19             objects = session.createQuery("select e from FriendshipCat
     e order by e.id", FriendshipCat.class)
20                     .getResultList();
21             session.getTransaction().commit();
22             session.close();
23
24             return objects;
25         } catch (HibernateException e) {
26             throw new DAOException(e.getMessage(), e);
27         }
28     }
29
30     @Override
31     public boolean refract(FriendshipCat object) throws DAOException {
32         try {
33             Session session = HibernateUtil.getSessionFactory().
    openSession();
34             session.getTransaction().begin();
35             session.update(object);
36             session.getTransaction().commit();
37             session.close();
38
39             return true;
40         } catch (HibernateException e) {
41             throw new DAOException(e.getMessage(), e);
42         }
43     }
44
45     @Override
46     public boolean add(FriendshipCat object) throws DAOException {
```

```
47          try {
48              Session session = HibernateUtil.getSessionFactory().
    openSession();
49              session.getTransaction().begin();
50              session.save(object);
51              session.getTransaction().commit();
52              session.close();
53
54              return true;
55          } catch (HibernateException e) {
56              throw new DAOException(e.getMessage(), e);
57          }
58      }
59
60      @Override
61      public boolean del(FriendshipCat object) throws DAOException {
62          try {
63              Session session = HibernateUtil.getSessionFactory().
    openSession();
64              session.getTransaction().begin();
65              session.delete(object);
66              session.getTransaction().commit();
67              session.close();
68
69              return true;
70          } catch (HibernateException e) {
71              throw new DAOException(e.getMessage(), e);
72          }
73      }
74
75      @Override
76      public FriendshipCat getById(long id) throws DAOException {
77          try {
78              Session session = HibernateUtil.getSessionFactory().
    openSession();
79              FriendshipCat item = session.byId(FriendshipCat.class).
    load(id);
80              session.close();
81
82              return item;
83          } catch (HibernateException e) {
84              throw new DAOException(e.getMessage(), e);
85          }
86      }
87  }
```

Листинг 1.20: OwnersDAO.java

```
1  package dao.daoImpl;
2
3  import dao.daoInterface.DAO;
4  import dao.entities.Owners;
5  import dao.tools.DAOException;
6  import org.hibernate.HibernateException;
7  import org.hibernate.Session;
8  import dao.tools.HibernateUtil;
9
10 import java.util.List;
11
12 public class OwnersDAO implements DAO<Owners> {
13
14     @Override
15     public List<Owners> findAll() throws DAOException {
16         try {
17             List<Owners> objects;
18             Session session = HibernateUtil.getSessionFactory().
    openSession();
19             session.getTransaction().begin();
20             objects = session.createQuery("select e from Owners e
    order by e.id", Owners.class)
21                     .getResultList();
22             session.getTransaction().commit();
23             session.close();
24
25             return objects;
26         } catch (HibernateException e) {
27             throw new DAOException(e.getMessage(), e);
28         }
29     }
30
31     @Override
32     public boolean refract(Owners object) throws DAOException {
33         try {
34             Session session = HibernateUtil.getSessionFactory().
    openSession();
35             session.getTransaction().begin();
36             session.update(object);
37             session.getTransaction().commit();
38             session.close();
39
40             return true;
41         } catch (HibernateException e) {
42             throw new DAOException(e.getMessage(), e);
43         }
44     }
45
46     @Override
```

```
47      public boolean add(Owners object) throws DAOException {
48          try {
49              Session session = HibernateUtil.getSessionFactory().
    openSession();
50              session.getTransaction().begin();
51              session.save(object);
52              session.getTransaction().commit();
53              session.close();
54
55              return true;
56          } catch (HibernateException e) {
57              throw new DAOException(e.getMessage(), e);
58          }
59      }
60
61      @Override
62      public boolean del(Owners object) throws DAOException {
63          try {
64              Session session = HibernateUtil.getSessionFactory().
    openSession();
65              session.getTransaction().begin();
66              session.delete(object);
67              session.getTransaction().commit();
68              session.close();
69
70              return true;
71          } catch (HibernateException e) {
72              throw new DAOException(e.getMessage(), e);
73          }
74      }
75
76      @Override
77      public Owners getById(long id) throws DAOException {
78          try {
79              Session session = HibernateUtil.getSessionFactory().
    openSession();
80              Owners item = session.byId(Owners.class).load(id);
81              session.close();
82
83              return item;
84          } catch (HibernateException e) {
85              throw new DAOException(e.getMessage(), e);
86          }
87      }
88  }
```

Листинг 1.21: OwnershipCatsDAO.java

```
1  package dao.daoImpl;
2
3  import dao.daoInterface.DAO;
4  import dao.entities.OwnershipCat;
5  import dao.tools.DAOException;
6  import dao.tools.HibernateUtil;
7  import org.hibernate.HibernateException;
8  import org.hibernate.Session;
9
10 import java.util.List;
11
12 public class OwnershipCatsDAO implements DAO<OwnershipCat> {
13     @Override
14     public List<OwnershipCat> findAll() throws DAOException {
15         try {
16             List<OwnershipCat> objects;
17             Session session = HibernateUtil.getSessionFactory().
    openSession();
18             session.getTransaction().begin();
19             objects = session.createQuery("select e from OwnershipCat
    e order by e.id", OwnershipCat.class)
20                     .getResultList();
21             session.getTransaction().commit();
22             session.close();
23
24             return objects;
25         } catch (HibernateException e) {
26             throw new DAOException(e.getMessage(), e);
27         }
28     }
29
30     @Override
31     public boolean refract(OwnershipCat object) throws DAOException {
32         Session session = HibernateUtil.getSessionFactory().
    openSession();
33         session.getTransaction().begin();
34         session.update(object);
35         session.getTransaction().commit();
36         session.close();
37         return true;
38     }
39
40     @Override
41     public boolean add(OwnershipCat object) throws DAOException {
42         try {
43             Session session = HibernateUtil.getSessionFactory().
    openSession();
44             session.getTransaction().begin();
45             session.save(object);
```

```
46              session.getTransaction().commit();
47              session.close();
48
49              return true;
50          } catch (HibernateException e) {
51              throw new DAOException(e.getMessage(), e);
52          }
53      }
54
55      @Override
56      public boolean del(OwnershipCat object) throws DAOException {
57          try {
58              Session session = HibernateUtil.getSessionFactory().
    openSession();
59              session.getTransaction().begin();
60              session.delete(object);
61              session.getTransaction().commit();
62              session.close();
63
64              return true;
65          } catch (HibernateException e) {
66              throw new DAOException(e.getMessage(), e);
67          }
68      }
69
70      @Override
71      public OwnershipCat getById(long id) throws DAOException {
72          try {
73              Session session = HibernateUtil.getSessionFactory().
    openSession();
74              OwnershipCat item = session.byId(OwnershipCat.class).load(
    id);
75              session.close();
76
77              return item;
78          } catch (HibernateException e) {
79              throw new DAOException(e.getMessage(), e);
80          }
81      }
82  }
```

Листинг 1.22: DAO.java

```
1  package dao.daoInterface;
2
3  import dao.tools.DAOException;
4
5  import java.util.List;
6
7  public interface DAO<T> {
8      List<T> findAll() throws DAOException;
9
10     boolean refract(T object) throws DAOException;
11
12     boolean add(T object) throws DAOException;
13
14     boolean del(T object) throws DAOException;
15
16     T getById(long id) throws DAOException;
17 }
```

Листинг 1.23: Cat.java

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

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

Листинг 1.24: FriendshipCat.java

```
1  package dao.entities;
2
3  import javax.persistence.*;
4  import java.util.Objects;
5
6  @Entity
7  @Table(name = "friendshipcats")
8  public class FriendshipCat {
9      @GeneratedValue(strategy = GenerationType.IDENTITY)
10     @Id
11     @Column(name = "id", nullable = false)
12     private long id;
13     @Basic
14     @Column(name = "first_cat_id", nullable = false)
15     private long firstCatId;
16     @Basic
17     @Column(name = "second_cat_id", nullable = false)
18     private long secondCatId;
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
44     @Override
45     public boolean equals(Object o) {
46         if (this == o) return true;
47         if (o == null || getClass() != o.getClass()) return false;
48         FriendshipCat that = (FriendshipCat) o;
49         return id == that.id && firstCatId == that.firstCatId &&
```

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

Листинг 1.25: Owners.java

```
1  package dao.entities;
2
3  import javax.persistence.*;
4  import java.sql.Timestamp;
5  import java.util.Objects;
6
7  @Entity
8  @Table(name = "owners")
9  public class Owners {
10     @GeneratedValue(strategy = GenerationType.IDENTITY)
11     @Id
12     @Column(name = "id", nullable = false)
13     private long id;
14     @Basic
15     @Column(name = "name", nullable = true, length = -1)
16     private String name;
17     @Basic
18     @Column(name = "birthday", nullable = true)
19     private Timestamp birthday;
20
21     public long getId() {
22         return id;
23     }
24
25     public void setId(long id) {
26         this.id = id;
27     }
28
29     public String getName() {
30         return name;
31     }
32
33     public void setName(String name) {
34         this.name = name;
35     }
36
37     public Timestamp getBirthday() {
38         return birthday;
39     }
40
41     public void setBirthday(Timestamp birthday) {
42         this.birthday = birthday;
43     }
44
45     @Override
46     public boolean equals(Object o) {
47         if (this == o) return true;
48         if (o == null || getClass() != o.getClass()) return false;
49         Owners owners = (Owners) o;
```

```
50          return id == owners.id && Objects.equals(name, owners.name) &&
        Objects.equals(birthday, owners.birthday);
51      }
52
53      @Override
54      public int hashCode() {
55          return Objects.hash(id, name, birthday);
56      }
57  }
```

Листинг 1.26: OwnershipCat.java

```
package dao.entities;

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

@Entity
@Table(name = "ownershipCats")
public class OwnershipCat {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id", nullable = false)
    private long id;
    @Basic
    @Column(name = "owner_id", nullable = false)
    private long ownerId;
    @Basic
    @Column(name = "cat_id", nullable = false)
    private long catId;

    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;
    }

    public void setCatId(long catId) {
        this.catId = catId;
    }

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

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

Листинг 1.27: Colors.java

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

Листинг 1.28: DAOException.java

```
1  package 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 1.29: HibernateUtil.java

```
package dao.tools;

import org.hibernate.SessionFactory;

import java.io.File;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory =
   initSessionFactory();

    private static SessionFactory initSessionFactory() {
        try {
            return new Configuration().configure(
                    new File("C:\\Users\\HTMLD\\Documents\\GitHub\\" +
                            "ferbatorTeh\\kotiki-java\\src\\main\\
   resources\\" +
                            "hibernate.cfg.xml"))
                    .buildSessionFactory();
        } catch (Throwable ex) {
            System.err.println("Initial SessionFactory creation failed
   ." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        if (sessionFactory == null) {
            initSessionFactory();
        }

        return sessionFactory;
    }

    public static void close() {
        getSessionFactory().close();
    }
}
```

Листинг 1.30: ShelterService.java

```
1  package services;
2
3  import dao.daoInterface.DAO;
4  import dao.entities.Cat;
5  import dao.entities.FriendshipCat;
6  import dao.entities.Owners;
7  import dao.entities.OwnershipCat;
8  import dao.enums.Colors;
9  import dao.tools.DAOException;
10 import services.tools.ShelterServiceException;
11
12 import java.sql.Timestamp;
13 import java.util.ArrayList;
14 import java.util.List;
15
16 public class ShelterService {
17     private DAO<Owners> daoOwn;
18     private DAO<Cat> daoCat;
19     private DAO<OwnershipCat> daoOwnShip;
20     private DAO<FriendshipCat> daoFriendShip;
21
22     public ShelterService(DAO<Owners> daoOwn, DAO<Cat> daoCat, DAO<
       OwnershipCat> daoOwnShip, DAO<FriendshipCat> daoFriendShip) {
23         this.daoOwn = daoOwn;
24         this.daoCat = daoCat;
25         this.daoOwnShip = daoOwnShip;
26         this.daoFriendShip = daoFriendShip;
27     }
28
29     public boolean addOwnerToBase(String name, String birthday) throws
       ShelterServiceException {
30         Owners own = new Owners();
31         own.setName(name);
32         own.setBirthday(Timestamp.valueOf(birthday));
33         try {
34             return daoOwn.add(own);
35         } catch (DAOException e) {
36             throw new ShelterServiceException("Error when adding", e);
37         }
38     }
39
40     public boolean addCatToBase(String name, Colors color, String
       breed, String birthday) throws ShelterServiceException {
41         Cat cat = new Cat();
42         cat.setName(name);
43         cat.setColor(color);
44         cat.setBirthday(Timestamp.valueOf(birthday));
45         cat.setBreed(breed);
46         try {
```

```
47              return daoCat.add(cat);
48          } catch (DAOException e) {
49              throw new ShelterServiceException("Error when adding", e);
50          }
51      }
52
53      public boolean delOwnerFromBase(long idOwn) throws
    ShelterServiceException {
54          List<OwnershipCat> tmpOwnershipCats = null;
55          try {
56              tmpOwnershipCats = daoOwnShip.findAll();
57          } catch (DAOException e) {
58              throw new ShelterServiceException("Error when findAll", e)
    ;
59          }
60          for (OwnershipCat ship : tmpOwnershipCats) {
61              if (ship.getOwnerId() == idOwn)
62                  try {
63                      return daoOwnShip.del(ship);
64                  } catch (DAOException e) {
65                      throw new ShelterServiceException("Error when
    adding", e);
66                  }
67          }
68          try {
69              return daoOwn.del(daoOwn.getById(idOwn));
70          } catch (DAOException e) {
71              throw new ShelterServiceException("Error when adding", e);
72          }
73      }
74
75      public boolean delCatFromBase(long idCat) throws
    ShelterServiceException {
76          List<OwnershipCat> tmpOwnershipCats = null;
77          try {
78              tmpOwnershipCats = daoOwnShip.findAll();
79          } catch (DAOException e) {
80              throw new ShelterServiceException("Error when findAll", e)
    ;
81          }
82          for (OwnershipCat ship : tmpOwnershipCats) {
83              if (ship.getCatId() == idCat)
84                  try {
85                      return daoOwnShip.del(ship);
86                  } catch (DAOException e) {
87                      throw new ShelterServiceException("Error when
    delete", e);
88                  }
89          }
90          List<FriendshipCat> tmpFriendshipCats = null;
```

```
91          try {
92              tmpFriendshipCats = daoFriendShip.findAll();
93          } catch (DAOException e) {
94              throw new ShelterServiceException("Error when findAll", e)
    ;
95          }
96          for (FriendshipCat ship : tmpFriendshipCats) {
97              if (ship.getFirstCatId() == idCat || ship.getSecondCatId()
     == idCat)
98                  try {
99                      return daoFriendShip.del(ship);
100                 } catch (DAOException e) {
101                     throw new ShelterServiceException("Error when
    delete", e);
102                 }
103         }
104         try {
105             return daoCat.del(daoCat.getById(idCat));
106         } catch (DAOException e) {
107             throw new ShelterServiceException("Error when delete", e);
108         }
109     }
110
111     public boolean PŸstartatOwnership(long idOwner, long idCat) throws
     ShelterServiceException {
112         OwnershipCat ship = new OwnershipCat();
113         ship.setOwnerId(idOwner);
114         ship.setCatId(idCat);
115         try {
116             return daoOwnShip.add(ship);
117         } catch (DAOException e) {
118             throw new ShelterServiceException("Error when adding", e);
119         }
120     }
121
122     public boolean cancelCatOwnership(long idOwner, long idCat) throws
     ShelterServiceException {
123         List<OwnershipCat> tmpOwnershipCats = null;
124         try {
125             tmpOwnershipCats = daoOwnShip.findAll();
126         } catch (DAOException e) {
127             throw new ShelterServiceException("Error when findAll", e)
    ;
128         }
129         for (OwnershipCat ship : tmpOwnershipCats) {
130             if (ship.getOwnerId() == idOwner && ship.getCatId() ==
    idCat)
131                 try {
132                     return daoOwnShip.del(ship);
133                 } catch (DAOException e) {
```

```
134                     throw new ShelterServiceException("Error when
    delete", e);
135                 }
136         }
137         return false;
138     }
139
140     public boolean PŮstartatFriendship(long idFirstCat, long
    idSecondCat) throws ShelterServiceException {
141         FriendshipCat ship = new FriendshipCat();
142         ship.setFirstCatId(idFirstCat);
143         ship.setSecondCatId(idSecondCat);
144         try {
145             return daoFriendShip.add(ship);
146         } catch (DAOException e) {
147             throw new ShelterServiceException("Error when adding", e);
148         }
149     }
150
151     public boolean cancelCatFriendship(long idFirstCat, long
    idSecondCat) throws ShelterServiceException {
152         List<FriendshipCat> tmpFriendshipCats = null;
153         try {
154             tmpFriendshipCats = daoFriendShip.findAll();
155         } catch (DAOException e) {
156             throw new ShelterServiceException("Error when findAll", e)
    ;
157         }
158         for (FriendshipCat ship : tmpFriendshipCats) {
159             if (ship.getFirstCatId() == idFirstCat && ship.
    getSecondCatId() == idSecondCat)
160                 try {
161                     return daoFriendShip.del(ship);
162                 } catch (DAOException e) {
163                     throw new ShelterServiceException("Error when
    delete", e);
164                 }
165         }
166         return false;
167     }
168
169     public List<Cat> getListFriendsForCat(long id) throws
    ShelterServiceException {
170         List<FriendshipCat> tmpFriendshipCats = null;
171         try {
172             tmpFriendshipCats = daoFriendShip.findAll();
173         } catch (DAOException e) {
174             throw new ShelterServiceException("Error when findAll", e)
    ;
175         }
```

```
176         List<Cat> tmpFriendsForCat = new ArrayList<>();
177         for (FriendshipCat ship : tmpFriendshipCats) {
178             if (ship.getFirstCatId() == id)
179                 try {
180                     tmpFriendsForCat.add(daoCat.getById(ship.
    getSecondCatId()));
181                 } catch (DAOException e) {
182                     throw new ShelterServiceException("Error when
    adding", e);
183                 }
184             if (ship.getSecondCatId() == id) {
185                 try {
186                     tmpFriendsForCat.add(daoCat.getById(ship.
    getFirstCatId()));
187                 } catch (DAOException e) {
188                     throw new ShelterServiceException("Error when
    adding", e);
189                 }
190             }
191         }
192
193         return tmpFriendsForCat;
194      }
195
196      public List<Cat> getListCatsForOwner(long id) throws
    ShelterServiceException {
197         List<OwnershipCat> tmpOwnershipCats = null;
198         try {
199             tmpOwnershipCats = daoOwnShip.findAll();
200         } catch (dao.tools.DAOException e) {
201             throw new ShelterServiceException("Error when findAll", e)
    ;
202         }
203         List<Cat> tmpCatForOwner = new ArrayList<>();
204         for (OwnershipCat ship : tmpOwnershipCats) {
205             if (ship.getOwnerId() == id)
206                 try {
207                     tmpCatForOwner.add(daoCat.getById(ship.getCatId())
    );
208                 } catch (dao.tools.DAOException e) {
209                     throw new ShelterServiceException("Error when
    adding", e);
210                 }
211         }
212         return tmpCatForOwner;
213      }
214 }
```

Листинг 1.31: ShelterServiceException.java

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

Листинг 1.32: ShelterServiceTest.java

```
1  package services;
2  import dao.daoInterface.DAO;
3  import dao.entities.Cat;
4  import dao.entities.FriendshipCat;
5  import dao.entities.Owners;
6  import dao.entities.OwnershipCat;
7  import dao.enums.Colors;
8  import org.junit.jupiter.api.BeforeEach;
9  import org.junit.jupiter.api.Test;
10 import org.mockito.Mock;
11 import org.mockito.MockitoAnnotations;
12 import services.tools.ShelterServiceException;
13
14 import java.sql.Timestamp;
15 import java.util.ArrayList;
16 import java.util.List;
17
18 import static org.junit.jupiter.api.Assertions.assertTrue;
19 import static org.mockito.Mockito.when;
20
21 class ShelterServiceTest {
22     @Mock
23     private DAO<Owners> daoOwn;
24     @Mock
25     private DAO<Cat> daoCat;
26     @Mock
27     private DAO<OwnershipCat> daoOwnShip;
28     @Mock
29     private DAO<FriendshipCat> daoFriendShip;
30
31     private Owners own;
32     private Cat cat1;
33     private Cat cat2;
34     private OwnershipCat ownAndCat1;
35     private OwnershipCat ownAndCat2;
36     private FriendshipCat cat1AndCat2;
37
38     private ShelterService service;
39
40     @BeforeEach
41     void setUp() {
42         own = new Owners();
43         own.setName("sock");
44         own.setBirthday(Timestamp.valueOf("2002-01-12 00:00:00"));
45
46         cat1 = new Cat();
47         cat1.setName("Boris");
48         cat1.setColor(Colors.Black);
49         cat1.setBirthday(Timestamp.valueOf("2002-01-12 00:00:00"));
```

```
50          cat1.setBreed("breed");
51
52          cat2 = new Cat();
53          cat2.setName("Stepan");
54          cat2.setColor(Colors.Orange);
55          cat2.setBirthday(Timestamp.valueOf("2002-01-12 00:00:00"));
56          cat2.setBreed("breed");
57
58          ownAndCat1 = new OwnershipCat();
59          ownAndCat1.setOwnerId(1);
60          ownAndCat1.setCatId(1);
61
62          ownAndCat2 = new OwnershipCat();
63          ownAndCat2.setOwnerId(1);
64          ownAndCat2.setCatId(2);
65
66          cat1AndCat2 = new FriendshipCat();
67          cat1AndCat2.setFirstCatId(1);
68          cat1AndCat2.setSecondCatId(2);
69
70      }
71
72      ShelterServiceTest() {
73          MockitoAnnotations.initMocks(this);
74          this.service = new ShelterService(daoOwn, daoCat, daoOwnShip,
    daoFriendShip);
75      }
76
77      @Test
78      void addOwnerToBase() throws ShelterServiceException {
79          try {
80              when(daoOwn.add(own)).thenReturn(true);
81          } catch (dao.tools.DAOException e) {
82              e.printStackTrace();
83          }
84          boolean test = service.addOwnerToBase("sock", "2002-01-12
    00:00:00");
85          assertTrue(test);
86      }
87
88      @Test
89      void addCatToBase() throws ShelterServiceException {
90          try {
91              when(daoCat.add(cat1)).thenReturn(true);
92          } catch (dao.tools.DAOException e) {
93              e.printStackTrace();
94          }
95          boolean test = service.addCatToBase("Boris", Colors.Black,"
    breed","2002-01-12 00:00:00");
96          assertTrue(test);
```

```
97      }
98
99      @Test
100     void delOwnerFromBase() throws ShelterServiceException {
101         try {
102             when(daoOwnShip.del(ownAndCat1)).thenReturn(true);
103         } catch (dao.tools.DAOException e) {
104             e.printStackTrace();
105         }
106         List<OwnershipCat> ownershipCats = new ArrayList<>();
107         ownershipCats.add(ownAndCat1);
108         try {
109             when(daoOwnShip.findAll()).thenReturn(ownershipCats);
110         } catch (dao.tools.DAOException e) {
111             e.printStackTrace();
112         }
113         try {
114             when(daoOwn.getById(1)).thenReturn(own);
115         } catch (dao.tools.DAOException e) {
116             e.printStackTrace();
117         }
118         try {
119             when(daoOwn.del(own)).thenReturn(true);
120         } catch (dao.tools.DAOException e) {
121             e.printStackTrace();
122         }
123         boolean test = service.delOwnerFromBase(1);
124         assertTrue(test);
125     }
126
127     @Test
128     void delCatFromBase() throws ShelterServiceException {
129         try {
130             when(daoOwnShip.del(ownAndCat1)).thenReturn(true);
131         } catch (dao.tools.DAOException e) {
132             e.printStackTrace();
133         }
134         List<OwnershipCat> ownershipCats = new ArrayList<>();
135         ownershipCats.add(ownAndCat1);
136         try {
137             when(daoOwnShip.findAll()).thenReturn(ownershipCats);
138         } catch (dao.tools.DAOException e) {
139             e.printStackTrace();
140         }
141         try {
142             when(daoCat.getById(1)).thenReturn(cat1);
143         } catch (dao.tools.DAOException e) {
144             e.printStackTrace();
145         }
146         try {
```

```
147             when(daoCat.del(cat1)).thenReturn(true);
148         } catch (dao.tools.DAOException e) {
149             e.printStackTrace();
150         }
151         List<FriendshipCat> friendshipCats = new ArrayList<>();
152         friendshipCats.add(cat1AndCat2);
153         try {
154             when(daoFriendShip.findAll()).thenReturn(friendshipCats);
155         } catch (dao.tools.DAOException e) {
156             e.printStackTrace();
157         }
158         boolean test = service.delCatFromBase(1);
159         assertTrue(test);
160     }
161 
162     @Test
163     void PŭstartatOwnership() throws ShelterServiceException {
164         try {
165             when(daoOwnShip.add(ownAndCat1)).thenReturn(true);
166         } catch (dao.tools.DAOException e) {
167             e.printStackTrace();
168         }
169         boolean test = service.PŭstartatOwnership(1,1);
170         assertTrue(test);
171     }
172 
173     @Test
174     void cancelCatOwnership() throws ShelterServiceException {
175         try {
176             when(daoOwnShip.del(ownAndCat1)).thenReturn(true);
177         } catch (dao.tools.DAOException e) {
178             e.printStackTrace();
179         }
180         List<OwnershipCat> ownershipCats = new ArrayList<>();
181         ownershipCats.add(ownAndCat1);
182         try {
183             when(daoOwnShip.findAll()).thenReturn(ownershipCats);
184         } catch (dao.tools.DAOException e) {
185             e.printStackTrace();
186         }
187         boolean test = service.cancelCatOwnership(1,1);
188         assertTrue(test);
189     }
190 
191     @Test
192     void PŭstartatFriendship() throws ShelterServiceException {
193         try {
194             when(daoFriendShip.add(cat1AndCat2)).thenReturn(true);
195         } catch (dao.tools.DAOException e) {
196             e.printStackTrace();
```

```
197         }
198         boolean test = service.PŬstartatFriendship(1,2);
199         assertTrue(test);
200     }
201 
202     @Test
203     void cancelCatFriendship() throws ShelterServiceException {
204         try {
205             when(daoFriendShip.del(cat1AndCat2)).thenReturn(true);
206         } catch (dao.tools.DAOException e) {
207             e.printStackTrace();
208         }
209         List<FriendshipCat> friendshipCats = new ArrayList<>();
210         friendshipCats.add(cat1AndCat2);
211         try {
212             when(daoFriendShip.findAll()).thenReturn(friendshipCats);
213         } catch (dao.tools.DAOException e) {
214             e.printStackTrace();
215         }
216         boolean test = service.cancelCatFriendship(1,2);
217         assertTrue(test);
218     }
219 }
```