

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Лабораторная работа №2

Выполнил студент:

Колпикова Ксения Денисовна
группа: М32071

Проверил:

Чижишев 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: Main.java

```

1 package controller;
2
3 import DAO.enums.MyColors;
4 import DAO.implemetations.CatsDao;
5 import DAO.implemetations.FriendshipDao;
6 import DAO.implemetations.OwnersDao;
7 import DAO.implemetations.ShelterDao;
8 import DAO.interfaces.Dao;
9 import DAO.models.CatsEntity;
10 import DAO.models.CatsForOwnerEntity;
11 import DAO.models.FriendsForCatEntity;
12 import DAO.models.OwnersEntity;
13 import service.KotikiService;
14 import service.tools.KotikiException;
15
16 import java.sql.Timestamp;
17 import java.util.Objects;
18 import java.util.Scanner;
19
20 public class Main {
21     public static void main(String args[]) throws KotikiException {
22         Dao<CatsEntity> catsDao = new CatsDao();
23         Dao<OwnersEntity> ownerDao = new OwnersDao();
24         Dao<FriendsForCatEntity> friendshipDao = new FriendshipDao();
25         Dao<CatsForOwnerEntity> shelterDao = new ShelterDao();
26         KotikiService kotiki = new KotikiService(catsDao, ownerDao,
27             friendshipDao, shelterDao);
28
29         Scanner in = new Scanner(System.in);
30         String str = null;
31
32         while (!Objects.equals(str, "exit")) {
33             System.out.print("P'PIPμPrPëC,Pμ PsPiC†PëCтн : " + "\n");
34             System.out.print("P"PsP±P°PIPëC,Ch PePsC,P° – 1" + "\n");
35             System.out.print("P"PsP±P°PIPëC,Ch C...PsP.CЇPëPSP° – 2" + "\n");
36             System.out.print("PμCтPëC†PIPsPëC,Ch PePsC,P° C...PsP.CЇPëPSCr – 3" + "\n");
37             System.out.print("PμPsPrCтCrP¶PëC,Ch PePsC,PsPI – 4" + "\n");
38             System.out.print("PJPrP°P»PëC,Ch PePsC,P° – 5" + "\n");
39             System.out.print("PJPrP°P»PëC,Ch C...PsP.CЇPëPSP° – 6" + "\n");
40             System.out.print("P P°P.PsCтPIP°C,Ch PrCтCrP¶P±Cr PePsC,PsPI – 7" + "\n");
41             System.out.print("PŸPrPμP»P°C,Ch PePsC,P° P±PμP.C...PsP.CЇPëPSP° – 8" + "\n");
42             System.out.print("P'C<C...PsPr PI PiP»P°PIPSPsPμ PjPμPSCTн – exit" + "\n");

```

```

43         str = in.nextLine();
44         switch (str) {
45             case "1":
46                 CatsEntity cat = new CatsEntity();
47                 System.out.print("P'PIPμPrPëC,Pμ PëPjC¼ PePsC,P°" + "\n");
48                 String name = in.nextLine();
49                 System.out.print("P'PIPμPrPëC,Pμ PiPsCᵀPsPrCᶠ PePsC,P°" + "\n");
50                 String breed = in.nextLine();
51                 System.out.print("P'PIPμPrPëC,Pμ PrP°C,Cᶠ CᵀPsP¶PrPμPSPëC¼ PePsC,P°" + "\n");
52                 System.out.print("P' PIPëPrPμ PiPiPiPiPjPjPrPr— " + "\n");
53                 Timestamp birthday = Timestamp.valueOf(in.nextLine() + " 00:00:00");
54                 System.out.print("P'PIPμPrPëC,Pμ CᶠPIPμC, PePsC,P°: " + "\n");
55                 System.out.print("Black" + "\n");
56                 System.out.print("White" + "\n");
57                 System.out.print("Orange" + "\n");
58                 System.out.print("Brown" + "\n");
59                 MyColors color = null;
60                 switch (in.nextLine()) {
61                     case "Black":
62                         color = MyColors.Black;
63                         break;
64                     case "White":
65                         color = MyColors.White;
66                         break;
67                     case "Orange":
68                         color = MyColors.Orange;
69                     case "Brown":
70                         color = MyColors.Brown;
71                         break;
72                 }
73                 kotiki.addCat(name, birthday, color);
74                 break;
75             case "2":
76                 OwnersEntity owner = new OwnersEntity();
77                 System.out.print("P'PIPμPrPëC,Pμ PëPjC¼ C...PsP.C¼PëPSP°" + "\n");
78                 String nameOwner = in.nextLine();
79                 System.out.print("P'PIPμPrPëC,Pμ PrP°C,Cᶠ CᵀPsP¶PrPμPSPëC¼ C...PsP.C¼PëPSP°" + "\n");
80                 System.out.print("P' PIPëPrPμ PiPiPiPiPjPjPrPr— CᶠCᶠPjPjCᶠCᶠ:: " + "\n");
81                 Timestamp birthdayOwner = Timestamp.valueOf(in.nextLine());
82                 kotiki.addOwner(nameOwner, birthdayOwner);

```

```

83         break;
84     case "3":
85         System.out.print("P'PIPμPrPëC,Pμ id C...PsP·C¼PëPSP°"
+ "\n");
86         long idOwner = in.nextLong();
87         System.out.print("P'PIPμPrPëC,Pμ id PePsC,P°" + "\n");
88         long idCat = in.nextLong();
89         kotiki.shelterCat(idOwner, idCat);
90         break;
91     case "4":
92         System.out.print("P'PIPμPrPëC,Pμ id PePsC,P°" + "\n");
93         long id1 = in.nextLong();
94         System.out.print("P'PIPμPrPëC,Pμ id PrCTḂCrPiP°" + "\n
");
95         long id2 = in.nextLong();
96         kotiki.friendship(id1, id2);
97         break;
98     case "5":
99         System.out.print("P'PIPμPrPëC,Pμ id PePsC,P°" + "\n");
100        long id3 = in.nextLong();
101        kotiki.deleteCat(id3);
102        break;
103    case "6":
104        System.out.print("P'PIPμPrPëC,Pμ id C...PsP·C¼PëPSP°"
+ "\n");
105        long id4 = in.nextLong();
106        kotiki.deleteOwner(id4);
107        break;
108    case "7":
109        System.out.print("P'PIPμPrPëC,Pμ id PePsC,P°" + "\n");
110        long id5 = in.nextLong();
111        System.out.print("P'PIPμPrPëC,Pμ id PrCTḂCrPiP°" + "\n
");
112        long id6 = in.nextLong();
113        kotiki.deleteFriendship(id5, id6);
114        break;
115    case "8":
116        System.out.print("P'PIPμPrPëC,Pμ id C...PsP·C¼PëPSP°"
+ "\n");
117        long idOwner1 = in.nextLong();
118        System.out.print("P'PIPμPrPëC,Pμ id PePsC,P°" + "\n");
119        long idCat1 = in.nextLong();
120        kotiki.deleteShelter(idCat1, idOwner1);
121        break;
122    case "exit":
123        System.out.println("finish");
124        break;
125    }
126 }
127 }

```

128

129

}

Листинг 1.2: Bank.java

```

1 package models;
2
3 import services.EventManager;
4 import services.IAccount;
5 import tools.BanksException;
6
7 import java.util.ArrayList;
8 import java.util.List;
9
10 public class Bank {
11     public EventManager eventManager;
12     private List<Client> clients;
13     private List<IAccount> accountsOfClient;
14     public PercentOfDeposit PercentsOfDeposit;
15     public String nameOfBank;
16     public double commissionOfCreditOfBank;
17     public double limitOfCreditOfBank;
18     public double percentOfDebitOfBank;
19     public double limitationOfBank;
20
21     public Bank(String name, double commissionOfCredit, double
22 limitOfCredit, double percentOfDebit, double limitation,
23     PercentOfDeposit percentOfDeposit) throws
24 BanksException {
25         if (percentOfDeposit == null) throw new BanksException("
26 Invalid percents of deposit");
27         PercentsOfDeposit = percentOfDeposit;
28         if (name == null) throw new BanksException("Invalid name");
29         nameOfBank = name;
30         commissionOfCreditOfBank = commissionOfCredit;
31         limitOfCreditOfBank = limitOfCredit;
32         percentOfDebitOfBank = percentOfDebit;
33         clients = new ArrayList<>();
34         limitationOfBank = limitation;
35         accountsOfClient = new ArrayList<>();
36         eventManager = new EventManager("Change nameOfBank",
37             "Change commissionOfCreditOfBank", "Change
38 commissionOfCreditOfBank",
39             "Change limitOfCreditOfBank", "Change
40 percentOfDebitOfBank", "Change limitationOfBank");
41     }
42
43     public void addClientInBank(Client client) throws BanksException {
44         if (client == null) throw new BanksException("Invalid client")
45 ;
46         if (clients.stream().anyMatch(n -> n.nameOfClient == client.
47 nameOfClient) &&
48             clients.stream().anyMatch(n -> n.surnameOfBank ==
49 client.surnameOfBank))

```

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



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

Листинг 1.3: CentralBank.java

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

```

43     commissionOfCreditOfBank , bank.limitOfCreditOfBank , balance);
44         bank.addCreditInlAccount(credit);
45         return credit;
46
47     case "Debit":
48
49         Debit debit = new Debit(balance , bank.
50 percentOfDebitOfBank , countOfId++);
51         bank.addDebitInlAccount(debit);
52         return debit;
53
54     case "Deposit":
55
56         double depositPercent = 0;
57         int f = 0;
58         for ( Integer key : bank.PercentsOfDeposit.percents.
59 keySet() ) {
60             f++;
61             if (balance < key) {
62                 depositPercent = bank.PercentsOfDeposit.percents.
63                 get(key);
64             }
65             if( bank.PercentsOfDeposit.percents.keySet().size() -
66 f == 1 && depositPercent == 0)
67                 depositPercent = bank.PercentsOfDeposit.percents.
68                 get(key);
69         }
70         Deposit deposit = new Deposit(countOfId++, balance ,
71 time , depositPercent);
72         bank.addDepositInlAccount(deposit);
73         return deposit;
74     default:
75         throw new BanksException("Option not found");
76     }
77 }
78
79 public void refillMoneyOn(Bank bank , int balance , int id) throws
80 BanksException {
81     if (bank == null) throw new BanksException("Invalid Bank");
82     bank.getAccountId(id).refillOperation(balance);
83 }
84
85 public void withdrawalMoney(Bank bank , int balance , int id , Client
86 client) throws BanksException {
87     if (!checkClientPassportAndAddress(client) && balance > bank.
88 limitationOfBank && bank.isOperationInAccount(balance)) {
89         throw new BanksException("Invalid Withdrawal");
90     }
91     bank.getAccountId(id).withdrawalOperation(balance);
92 }

```

```
83
84     public void transferMoneyOnBalance(Bank bank1, int balance, int
85     id1, Bank bank2, int id2, Client client) throws BanksException {
86         if (!checkClientPassportAndAddress(client) && balance > bank1.
87         limitationOfBank && bank1.isOperationInAccount(balance)) {
88             throw new BanksException("Invalid Transfer");
89         }
90         bank1.getAccountById(id1).transferOperation(bank2.getAccountById(
91         id2), balance);
92     }
93
94     public void cancelRefill(Bank bank, int balance, int id) throws
95     BanksException {
96         IAccount operation = bank.getAccountById(id);
97         if (operation == null) throw new BanksException("Refill don't
98         exists");
99         operation.cancelRefillOperation(balance);
100     }
101
102     public void cancelTransfer(Bank bank1, int balance, int id1, Bank
103     bank2, int id2) throws BanksException {
104         IAccount operation = bank1.getAccountById(id1);
105         if (operation == null) throw new BanksException("Transfer don'
106         t exists");
107         operation.cancelTransferOperation(bank2.getAccountById(id2),
108         balance);
109     }
110
111     public void cancelWithdrawal(Bank bank, int balance, int id)
112     throws BanksException {
113         IAccount operation = bank.getAccountById(id);
114         if (operation == null) throw new BanksException("Withdrawal
115         don't exists");
116         operation.cancelWithdrawalOperation(balance);
117     }
118
119     private boolean checkClientPassportAndAddress(Client client) {
120         return !client.passportOfBank.isBlank() || !client.
121         passportOfBank.isBlank();
122     }
123
124     public int countBanks(){
125         return banks.size();
126     }
127 }
```

Листинг 1.4: Client.java

```
1 package models;
2
3 import tools.BanksException;
4
5 public class Client {
6     private static int _id = 0;
7     public int Id = 0;
8     public String nameOfClient;
9     public String surnameOfBank;
10    public String addressOfBank;
11    public String passportOfBank;
12
13    public Client(String name, String surname, String address, String
14    passport) throws BanksException {
15        Id = _id++;
16        if(name == null) {
17            throw new BanksException("Invalid name");
18        }
19        nameOfClient = name;
20        if(surname == null) {
21            throw new BanksException("Invalid surname");
22        }
23        surnameOfBank = surname;
24        addressOfBank = address;
25        passportOfBank = passport;
26    }
27    public static void update(String event){
28        System.out.print(event);
29    }
}
```

Листинг 1.5: Credit.java

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

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

Листинг 1.6: Debit.java

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



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

Листинг 1.7: Deposit.java

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

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

Листинг 1.8: PercentOfDeposit.java

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

ЛИСТИНГ 1.9: ClientBuilder.java

```
1 package services;
2
3 import models.Client;
4 import tools.BanksException;
5
6 public class ClientBuilder
7 {
8     private String name;
9     private String passport;
10    private String surname;
11    private String address;
12
13    public ClientBuilder changeName(String name) throws BanksException
14    {
15        if(name == null) throw new BanksException("Invalid name");
16        this.name = name;
17        return this;
18    }
19
20    public ClientBuilder changeSurname(String surname) throws
21    BanksException {
22        if(surname == null) throw new BanksException("Invalid surname"
23    );
24        this.surname = surname;
25        return this;
26    }
27
28    public ClientBuilder changePassport(String passport) throws
29    BanksException {
30        if(passport == null) throw new BanksException("Invalid
31    passport");
32        this.passport = passport;
33        return this;
34    }
35
36    public ClientBuilder changeAddress(String address) throws
37    BanksException {
38        if(address == null) throw new BanksException("Invalid address"
39    );
40        this.address = address;
41        return this;
42    }
43
44    public Client create() throws BanksException {
45        return new Client(name, surname, address, passport);
46    }
47 }
```

Листинг 1.10: EventManager.java

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

Листинг 1.11: IAccount.java

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

Листинг 1.12: BanksException.java

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


Листинг 1.13: MyColors.java

```
1 package DAO.enums;  
2 public enum MyColors {  
3     White ,  
4     Brown ,  
5     Black ,  
6     Orange  
7 }
```

Листинг 1.14: CatsDao.java

```
1 package DAO.implemetations;
2
3 import DAO.interfaces.Dao;
4 import DAO.models.CatsEntity;
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 CatsDao implements Dao<CatsEntity> {
13     @Override
14     public List<CatsEntity> findAllEntity() throws DaoException {
15         try {
16             List<CatsEntity> entity;
17             Session session = HibernateUtil.getSessionFactory().
18 openSession();
19             session.getTransaction().begin();
20             entity = session.createQuery("select e from CatsEntity e",
21 CatsEntity.class)
22                 .getResultList();
23             session.getTransaction().commit();
24             session.close();
25             return entity;
26         } catch (HibernateException e) {
27             throw new DaoException(e.getMessage(), e);
28         }
29     }
30
31     @Override
32     public boolean changeEntity(CatsEntity entity) throws DaoException
33     {
34         try {
35             Session session = HibernateUtil.getSessionFactory().
36 openSession();
37             session.getTransaction().begin();
38             session.update(entity);
39             session.getTransaction().commit();
40             session.close();
41             return true;
42         } catch (HibernateException e) {
43             throw new DaoException(e.getMessage(), e);
44         }
45     }
46
47     @Override
48     public boolean addEntity(CatsEntity entity) throws DaoException {
49         try {
```

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

Листинг 1.15: FriendshipDao.java

```
1 package DAO.implemetations;
2
3 import DAO.interfaces.Dao;
4 import DAO.models.FriendsForCatEntity;
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 FriendshipDao implements Dao<FriendsForCatEntity> {
13     @Override
14     public List<FriendsForCatEntity> findAllEntity() throws
15     DaoException {
16         try {
17             List<FriendsForCatEntity> entity;
18             Session session = HibernateUtil.getSessionFactory().
19             openSession();
20             session.getTransaction().begin();
21             entity = session.createQuery("select e from
22             FriendsForCatEntity e",
23                                     FriendsForCatEntity.class)
24                                     .getResultList();
25             session.getTransaction().commit();
26             session.close();
27             return entity;
28         } catch (HibernateException e) {
29             throw new DaoException(e.getMessage(), e);
30         }
31     }
32
33     @Override
34     public boolean changeEntity(FriendsForCatEntity entity) throws
35     DaoException {
36         try {
37             Session session = HibernateUtil.getSessionFactory().
38             openSession();
39             session.getTransaction().begin();
40             session.update(entity);
41             session.getTransaction().commit();
42             session.close();
43             return true;
44         } catch (HibernateException e) {
45             throw new DaoException(e.getMessage(), e);
46         }
47     }
48
49     @Override
```

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

Листинг 1.16: OwnersDao.java

```
1 package DAO.implemetations;
2
3 import DAO.models.OwnersEntity;
4 import DAO.tools.DaoException;
5 import org.hibernate.HibernateException;
6 import org.hibernate.Session;
7 import DAO.interfaces.Dao;
8 import DAO.tools.HibernateUtil;
9
10 import java.util.List;
11
12 public class OwnersDao implements Dao<OwnersEntity> {
13     @Override
14     public List<OwnersEntity> findAllEntity() throws DaoException {
15         try {
16             List<OwnersEntity> entity;
17             Session session = HibernateUtil.getSessionFactory().
18 openSession();
19             session.getTransaction().begin();
20             entity = session.createQuery("select e from OwnersEntity e
21 ", OwnersEntity.class)
22                 .getResultList();
23             session.getTransaction().commit();
24             session.close();
25             return entity;
26         } catch (HibernateException e) {
27             throw new DaoException(e.getMessage(), e);
28         }
29     }
30
31     @Override
32     public boolean changeEntity(OwnersEntity entity) throws
33 DaoException {
34         try {
35             Session session = HibernateUtil.getSessionFactory().
36 openSession();
37             session.getTransaction().begin();
38             session.update(entity);
39             session.getTransaction().commit();
40             session.close();
41             return true;
42         } catch (HibernateException e) {
43             throw new DaoException(e.getMessage(), e);
44         }
45     }
46
47     @Override
48     public boolean addEntity(OwnersEntity entity) throws DaoException
49 {
```

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

Листинг 1.17: ShelterDao.java

```
1 package DAO.implemetations;
2
3 import DAO.interfaces.Dao;
4 import DAO.models.CatsForOwnerEntity;
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 ShelterDao implements Dao<CatsForOwnerEntity> {
13     @Override
14     public List<CatsForOwnerEntity> findAllEntity() throws
15     DaoException {
16         try {
17             List<CatsForOwnerEntity> entity;
18             Session session = HibernateUtil.getSessionFactory().
19             openSession();
20             session.getTransaction().begin();
21             entity = session.createQuery("select e from
22             CatsForOwnerEntity e",
23             CatsForOwnerEntity.class)
24             .getResultList();
25             session.getTransaction().commit();
26             session.close();
27             return entity;
28         } catch (HibernateException e) {
29             throw new DaoException(e.getMessage(), e);
30         }
31     }
32
33     @Override
34     public boolean changeEntity(CatsForOwnerEntity entity) throws
35     DaoException {
36         try {
37             Session session = HibernateUtil.getSessionFactory().
38             openSession();
39             session.getTransaction().begin();
40             session.update(entity);
41             session.getTransaction().commit();
42             session.close();
43             return true;
44         } catch (HibernateException e) {
45             throw new DaoException(e.getMessage(), e);
46         }
47     }
48
49     @Override
```



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

Листинг 1.18: Dao.java

```
1 package DAO.interfaces;  
2  
3 import DAO.tools.DaoException;  
4  
5 import java.util.List;  
6  
7 public interface Dao<T> {  
8     List<T> findAllEntity() throws DaoException;  
9  
10    boolean changeEntity(T entity) throws DaoException;  
11  
12    boolean addEntity(T entity) throws DaoException;  
13  
14    boolean deleteEntity(T entity) throws DaoException;  
15  
16    T getByld(long entity) throws DaoException;  
17 }
```

Листинг 1.19: CatsEntity.java

```
1 package DAO.models;
2
3 import DAO.enums.MyColors;
4
5 import javax.persistence.*;
6 import java.sql.Timestamp;
7 import java.util.Objects;
8
9 @Entity
10 @Table(name = "cats", schema = "public", catalog = "postgres")
11 public class CatsEntity {
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     @Id
14     @Column(name = "id", nullable = false)
15     private long id;
16     @Basic
17     @Column(name = "name", nullable = true, length = -1)
18     private String name;
19     @Basic
20     @Column(name = "birth", nullable = true)
21     private Timestamp birth;
22     @Basic
23     @Column(name = "breed", nullable = true, length = -1)
24     private String breed;
25     @Basic
26     @Column(name = "color", nullable = true, length = -1)
27     @Enumerated(EnumType.STRING)
28     private MyColors color;
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 name) {
43         this.name = name;
44     }
45
46     public Timestamp getBirth() {
47         return birth;
48     }
49 }
```

```
50 public void setBirth(Timestamp birth) {
51     this.birth = birth;
52 }
53
54 public String getBreed() {
55     return breed;
56 }
57
58 public void setBreed(String breed) {
59     this.breed = breed;
60 }
61
62 public MyColors getColor() {
63     return color;
64 }
65
66 public void setColor(MyColors 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     CatsEntity that = (CatsEntity) o;
75     return id == that.id && Objects.equals(name, that.name) &&
Objects.equals(birth, that.birth) && Objects.equals(breed, that.
breed) && Objects.equals(color, that.color);
76 }
77
78 @Override
79 public int hashCode() {
80     return Objects.hash(id, name, birth, breed, color);
81 }
82 }
```

Листинг 1.20: CatsForOwnerEntity.java

```
1 package DAO.models;
2
3 import javax.persistence.*;
4 import java.util.Objects;
5
6 @Entity
7 @Table(name = "catsForOwner", schema = "public", catalog = "postgres")
8 public class CatsForOwnerEntity {
9     @GeneratedValue(strategy = GenerationType.IDENTITY)
10    @Id
11    @Column(name = "id", nullable = false)
12    private long id;
13    @Basic
14    @Column(name = "idCat", nullable = false)
15    private long idCat;
16    @Basic
17    @Column(name = "idOwner", nullable = false)
18    private long idOwner;
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 getIdCat() {
29        return idCat;
30    }
31
32    public void setIdCat(long idCat) {
33        this.idCat = idCat;
34    }
35
36    public long getIdOwner() {
37        return idOwner;
38    }
39
40    public void setIdOwner(long idOwner) {
41        this.idOwner = idOwner;
42    }
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        CatsForOwnerEntity that = (CatsForOwnerEntity) o;
```

```
50         return id == that.id && idCat == that.idCat && idOwner == that
51         .idOwner;
52     }
53     @Override
54     public int hashCode() {
55         return Objects.hash(id, idCat, idOwner);
56     }
57 }
```

ЛИСТИНГ 1.21: FriendsForCatEntity.java

```
1 package DAO.models;
2
3 import javax.persistence.*;
4 import java.util.Objects;
5
6 @Entity
7 @Table(name = "friendsForCat", schema = "public", catalog = "postgres"
8 )
9 public class FriendsForCatEntity {
10     @GeneratedValue(strategy = GenerationType.IDENTITY)
11     @Id
12     @Column(name = "id", nullable = false)
13     private long id;
14     @Basic
15     @Column(name = "idFriend", nullable = false)
16     private long idFriend;
17     @Basic
18     @Column(name = "idCat", nullable = false)
19     private long idCat;
20
21     public long getId() {
22         return id;
23     }
24
25     public void setId(long id) {
26         this.id = id;
27     }
28
29     public long getIdFriend() {
30         return idFriend;
31     }
32
33     public void setIdFriend(long idFriend) {
34         this.idFriend = idFriend;
35     }
36
37     public long getIdCat() {
38         return idCat;
39     }
40
41     public void setIdCat(long idCat) {
42         this.idCat = idCat;
43     }
44
45     @Override
46     public boolean equals(Object o) {
47         if (this == o) return true;
48
```

```
49         if (o == null || getClass() != o.getClass()) return false;
50         FriendsForCatEntity that = (FriendsForCatEntity) o;
51         return id == that.id && idFriend == that.idFriend && idCat ==
that.idCat;
52     }
53
54     @Override
55     public int hashCode() {
56         return Objects.hash(id, idFriend, idCat);
57     }
58 }
```


ЛИСТИНГ 1.22: OwnersEntity.java

```
1 package DAO.models;
2
3 import javax.persistence.*;
4 import java.sql.Timestamp;
5 import java.util.Objects;
6
7 @Entity
8 @Table(name = "owners", schema = "public", catalog = "postgres")
9 public class OwnersEntity {
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 = "date", nullable = true)
19     private Timestamp date;
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 getDate() {
38         return date;
39     }
40
41     public void setDate(Timestamp date) {
42         this.date = date;
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         OwnersEntity that = (OwnersEntity) o;
```

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

Листинг 1.23: 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 }
16 }
```

Листинг 1.24: HibernateUtil.java

```
1 package DAO.tools;
2
3 import org.hibernate.SessionFactory;
4 import org.hibernate.cfg.Configuration;
5
6 import java.io.File;
7
8 public class HibernateUtil {
9     private static final SessionFactory sessionFactory =
10         initSessionFactory();
11
12     private static SessionFactory initSessionFactory() {
13         try {
14             return new Configuration().configure(new File("/Users/
15                 kisssusha/IdeaProjects/kisssusha/kotiki—java/src/main/resources/
16                 hibernate.cfg.xml")).buildSessionFactory();
17         }
18         catch (Throwable ex) {
19             System.err.println("Initial SessionFactory creation failed
20                 ." + ex);
21             throw new ExceptionInInitializerError(ex);
22         }
23     }
24
25     public static SessionFactory getSessionFactory() {
26
27         if (sessionFactory == null){
28             initSessionFactory();
29         }
30
31         return sessionFactory;
32     }
33
34     public static void close() {
35         getSessionFactory().close();
36     }
37 }
```

Листинг 1.25: KotikiService.java

```
1 package service;
2
3 import DAO.enums.MyColors;
4 import DAO.interfaces.Dao;
5 import DAO.models.CatsEntity;
6 import DAO.models.CatsForOwnerEntity;
7 import DAO.models.FriendsForCatEntity;
8 import DAO.models.OwnersEntity;
9 import DAO.tools.DaoException;
10 import service.tools.KotikiException;
11
12 import java.sql.Timestamp;
13
14 public class KotikiService {
15
16     private final Dao<CatsEntity> catsDao;
17     private final Dao<OwnersEntity> ownerDao;
18     private final Dao<FriendsForCatEntity> friendshipDao;
19     private final Dao<CatsForOwnerEntity> shelterDao;
20
21
22     public KotikiService(Dao<CatsEntity> catsDao, Dao<OwnersEntity>
ownerDao,
23                         Dao<FriendsForCatEntity> friendshipDao,
24                         Dao<CatsForOwnerEntity> shelterDao) {
25         this.catsDao = catsDao;
26         this.ownerDao = ownerDao;
27         this.friendshipDao = friendshipDao;
28         this.shelterDao = shelterDao;
29     }
30
31
32     public boolean deleteCat(long idCat) throws KotikiException {
33         if (idCat == 0) throw new KotikiException("Invalid cat");
34         try {
35             for (FriendsForCatEntity fr : friendshipDao.findAllEntity
36 ()) {
37                 if (fr.getIdCat() == idCat ||
38                     fr.getIdFriend() == idCat) friendshipDao.
39 deleteEntity(fr);
40             }
41         } catch (DaoException e) {
42             throw new KotikiException(e.getMessage());
43         }
44
45         try {
46             return catsDao.deleteEntity(catsDao.getByld(idCat));
47         } catch (DaoException e) {
48             throw new KotikiException(e.getMessage());
49         }
50     }
51 }
```

```
47     }
48 }
49
50 public boolean deleteOwner(long idOwner) throws KotikiException {
51     if (idOwner == 0) throw new KotikiException("Invalid owner");
52     try {
53         for (CatsForOwnerEntity fr : shelterDao.findAllEntity()) {
54             if (fr.getIdOwner() == idOwner) shelterDao.
deleteEntity(fr);
55         }
56     } catch (DAO.tools.DaoException e) {
57         throw new KotikiException(e.getMessage());
58     }
59     try {
60         return ownerDao.deleteEntity(ownerDao.getByld(idOwner));
61     } catch (DAO.tools.DaoException e) {
62         throw new KotikiException(e.getMessage());
63     }
64 }
65
66 public boolean changeCat(long idCat) throws KotikiException {
67     if (idCat == 0) throw new KotikiException("Invalid cat");
68     try {
69         return catsDao.changeEntity(catsDao.getByld(idCat));
70     } catch (DAO.tools.DaoException e) {
71         throw new KotikiException(e.getMessage());
72     }
73 }
74
75 public boolean changeOwner(long owner) throws KotikiException {
76     if (owner == 0) throw new KotikiException("Invalid owner");
77     try {
78         return ownerDao.changeEntity(ownerDao.getByld(owner));
79     } catch (DAO.tools.DaoException e) {
80         throw new KotikiException(e.getMessage());
81     }
82 }
83
84 public boolean addOwner(String name, Timestamp birthday) throws
KotikiException {
85     OwnersEntity owner = new OwnersEntity();
86     owner.setName(name);
87     owner.setDate(birthday);
88     try {
89         return ownerDao.addEntity(owner);
90     } catch (DAO.tools.DaoException e) {
91         throw new KotikiException(e.getMessage());
92     }
93 }
94 }
```

```
95     public boolean addCat(String name, Timestamp birthday, MyColors
color) throws KotikiException {
96         CatsEntity cat = new CatsEntity();
97         cat.setColor(color);
98         cat.setName(name);
99         cat.setBirth(birthday);
100        try {
101            return catsDao.addEntity(cat);
102        } catch (DAO.tools.DaoException e) {
103            throw new KotikiException(e.getMessage());
104        }
105    }
106
107    public boolean friendship(long cat1, long cat2) throws
KotikiException {
108        if (cat1 == 0) throw new KotikiException("Invalid cat1");
109        if (cat2 == 0) throw new KotikiException("Invalid cat2");
110        FriendsForCatEntity friends = new FriendsForCatEntity();
111        friends.setIdCat(cat1);
112        friends.setIdFriend(cat2);
113
114        try {
115            return friendshipDao.addEntity(friends);
116        } catch (DAO.tools.DaoException e) {
117            throw new KotikiException(e.getMessage());
118        }
119    }
120
121    public boolean shelterCat(long owner, long cat) throws
KotikiException {
122        if (cat == 0) throw new KotikiException("Invalid cat");
123        if (owner == 0) throw new KotikiException("Invalid owner");
124        try {
125            for (CatsForOwnerEntity pt : shelterDao.findAllEntity()) {
126                if (pt.getIdCat() == cat) return false;
127            }
128        } catch (DAO.tools.DaoException e) {
129            throw new KotikiException(e.getMessage());
130        }
131        CatsForOwnerEntity pets = new CatsForOwnerEntity();
132        pets.setIdCat(cat);
133        pets.setIdOwner(owner);
134
135        try {
136            return shelterDao.addEntity(pets);
137        } catch (DAO.tools.DaoException e) {
138            throw new KotikiException(e.getMessage());
139        }
140    }
141
```

```
142     public boolean deleteFriendship(long id1, long id2) throws
KotikiException {
143         if (id1 == id2) throw new KotikiException("Invalid friends");
144         try {
145             for (FriendsForCatEntity fr : friendshipDao.findAllEntity
146             ()) {
147                 if (fr.getIdCat() == id1 && fr.getIdFriend() == id2 ||
148                     fr.getIdFriend() == id1 && fr.getIdCat() ==
149                     id2) return friendshipDao.deleteEntity(fr);
150             }
151         } catch (DAO.tools.DaoException e) {
152             throw new KotikiException(e.getMessage());
153         }
154         return false;
155     }
156
157     public boolean deleteShelter(long idCat, long idOwner) throws
KotikiException {
158         try {
159             for (CatsForOwnerEntity fr : shelterDao.findAllEntity()) {
160                 if (fr.getIdCat() == idCat && fr.getIdOwner() ==
161                 idOwner) shelterDao.deleteEntity(fr);
162             }
163         } catch (DAO.tools.DaoException e) {
164             throw new KotikiException(e.getMessage());
165         }
166         return true;
167     }
168 }
```


Листинг 1.26: KotikiException.java

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

ЛИСТИНГ 1.27: KotikiServiceTest.java

```
1 package service;
2
3 import DAO.enums.MyColors;
4 import DAO.interfaces.Dao;
5 import DAO.models.CatsEntity;
6 import DAO.models.CatsForOwnerEntity;
7 import DAO.models.FriendsForCatEntity;
8 import DAO.models.OwnersEntity;
9 import org.junit.jupiter.api.Test;
10 import org.mockito.Mock;
11 import org.mockito.MockitoAnnotations;
12 import service.tools.KotikiException;
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.*;
19 import static org.mockito.BDDMockito.*;
20
21 class KotikiServiceTest {
22     @Mock
23     private Dao<CatsEntity> catsDao;
24     @Mock
25     private Dao<OwnersEntity> ownerDao;
26     @Mock
27     private Dao<FriendsForCatEntity> friendshipDao;
28     @Mock
29     private Dao<CatsForOwnerEntity> shelterDao;
30     private KotikiService kotikiService;
31
32     public KotikiServiceTest() {
33         MockitoAnnotations.initMocks(this);
34         this.kotikiService = new KotikiService(catsDao, ownerDao,
35         friendshipDao, shelterDao);
36     }
37
38     @Test
39     void deleteCat() throws KotikiException {
40         CatsEntity cat = new CatsEntity();
41         FriendsForCatEntity friends = new FriendsForCatEntity();
42         List<FriendsForCatEntity> listFriends = new ArrayList<>();
43         try {
44             given(friendshipDao.findAllEntity()).willReturn(
45             listFriends);
46             } catch (DAO.tools.DaoException e) {
47                 e.printStackTrace();
48             }
49     }
```

```

48         try {
49             given(friendshipDao.deleteEntity(friends)).willReturn(true
);
50         } catch (DAO.tools.DaoException e) {
51             e.printStackTrace();
52         }
53         try {
54             given(catsDao.deleteEntity(cat)).willReturn(true);
55         } catch (DAO.tools.DaoException e) {
56             e.printStackTrace();
57         }
58         try {
59             given(catsDao.getByld(1)).willReturn(cat);
60         } catch (DAO.tools.DaoException e) {
61             e.printStackTrace();
62         }
63         boolean exist = kotikiService.deleteCat(1);
64         assertTrue(exist);
65     }
66
67     @Test
68     void addCat() throws KotikiException {
69         CatsEntity cat = new CatsEntity();
70         cat.setBirth(Timestamp.valueOf("2002-07-13 00:00:00"));
71         cat.setName("nice");
72         cat.setColor(MyColors.Black);
73         try {
74             given(catsDao.addEntity(cat)).willReturn(true);
75         } catch (DAO.tools.DaoException e) {
76             e.printStackTrace();
77         }
78         boolean exist = kotikiService.addCat("nice", Timestamp.valueOf
("2002-07-13 00:00:00"), MyColors.Black);
79         assertTrue(exist);
80     }
81
82     @Test
83     void friendship() throws KotikiException {
84         CatsEntity cat = new CatsEntity();
85         CatsEntity cat1 = new CatsEntity();
86         cat.setName("hfhf");
87         cat1.setName("hkh");
88         FriendsForCatEntity friends = new FriendsForCatEntity();
89         friends.setldCat(1);
90         friends.setldFriend(2);
91         try {
92             given(friendshipDao.addEntity(friends)).willReturn(true);
93         } catch (DAO.tools.DaoException e) {
94             e.printStackTrace();
95         }

```

```

96         boolean exist = kotikiService.friendship(1,2);
97         assertTrue(exist);
98     }
99
100     @Test
101     void shelterCat() throws KotikiException {
102         CatsEntity cat = new CatsEntity();
103         OwnersEntity own = new OwnersEntity();
104         CatsForOwnerEntity pets = new CatsForOwnerEntity();
105         pets.setIdCat(1);
106         pets.setOwnerId(2);
107         try {
108             given(shelterDao.addEntity(pets)).willReturn(true);
109         } catch (DAO.tools.DaoException e) {
110             e.printStackTrace();
111         }
112         boolean exist = kotikiService.shelterCat(2,1);
113         assertTrue(exist);
114     }
115
116     @Test
117     void deleteFriendship() throws KotikiException {
118
119         FriendsForCatEntity friends = new FriendsForCatEntity();
120         friends.setIdFriend(1);
121         friends.setIdCat(2);
122         List<FriendsForCatEntity> listFriends = new ArrayList<>();
123         listFriends.add(friends);
124         try {
125             given(friendshipDao.findAllEntity()).willReturn(
listFriends);
126         } catch (DAO.tools.DaoException e) {
127             e.printStackTrace();
128         }
129         try {
130             given(friendshipDao.deleteEntity(friends)).willReturn(true
);
131         } catch (DAO.tools.DaoException e) {
132             e.printStackTrace();
133         }
134         boolean exist = kotikiService.deleteFriendship(1,2);
135         assertTrue(exist);
136     }
137 }

```