

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Лабораторная работа №3

Выполнил студент:

Колпикова Ксения Денисовна
группа: М32071

Проверил:

Чижишев Константин Максимович

Санкт-Петербург,
2022 г.

1.1. Текст задания

3 лабораторная

К созданному в прошлой лабораторной сервису добавляется Spring.

Сервис должен предоставлять http интерфейс (REST API) для получения информации о конкретных котиках и владельцах и для получения фильтрованной информации (например, получить всех рыжих котиков)

Внимание: недопустимо отдавать через HTTP интерфейс сущности JPA. Рекомендуется создать отдельные оберточные классы.

Сервисы и dao должны превратиться в Spring Bean'ы с использованием Dependency Injection (Autowired). Dao при этом наследуют JpaRepository и имеет шаблонные Spring Data JPA методы: <https://www.baeldung.com/spring-data-repositoriesrepositories>

При сдаче лабораторной нужно будет показать работоспособность endpoint'ов через http запросы (рекомендуется Postman).

В рамках лабораторной к проекту должен быть добавлен CI/CD, запускающий тесты проекта kotiki-java.

1.2. Решение

Листинг 1.1: Main.java

```

1 import models.Bank;
2 import models.CentralBank;
3 import models.Client;
4 import models.PercentOfDeposit;
5 import services.ClientBuilder;
6 import tools.BanksException;
7
8 import java.util.Scanner;
9
10 public class Main {
11     public static void main(String[] args) throws BanksException {
12         CentralBank central = null;
13         Scanner in = new Scanner(System.in);
14         System.out.println("1 — PŸPsP·PrP°PSPëPµ PiPsP»PSPsPiPs
PeP»PëPµPSC,P° ");
15         System.out.println("2 — PŸPsP·PrP°PSPëPµ PeP»PëPµPSC,P° CÍ
PsPiCTbP°PSPëC†PµPSPëCŸPjPë");
16         System.out.println("Name, Surname, Address, Passport");
17         String str = in.nextLine();
18         ClientBuilder creation = null;
19         Client client = null;
20         Bank bank = null;
21         switch (str)
22         {
23             case "1":
24                 client = creation.changeName(in.nextLine()).
25                     changeSurname(in.nextLine()).changeAddress(in.
nextLine()).
26                     changePassport(in.nextLine()).create();
27                 break;
28             case "2":
29                 client = creation.changeName(in.nextLine()).
changeSurname(in.nextLine()).create();
30                 break;
31         }
32
33         while (str != "exit")
34         {
35             System.out.println("1 — PŸPsP·PrP°PSPëPµ P±P°PSPëP°");
36             System.out.println("2 — P'PsP±P°PIP»PµPSPëPµ PeP»PëPµPSC,P°
PI P±P°PSPëP°");
37             System.out.println("3 — PŸPsP·PrP°PSPëPµ CÍC†PµC,P° PI
P±P°PSPëPµ");
38             System.out.println("4 — PŸPSCŸC,PëPµ PrPµPSPµPi CÍPs
CÍC†PµC,P°");
39             System.out.println("5 — PµPsPiPsP»PSPµPSPëPµ CÍC†PµC,P°");
40             System.out.println("6 — PµPµCTbPµPIPsPr PrPµPSPµPi");
41             System.out.println("7 — PhC,PjPµPSP° PiPsPiPsP»PSPµPSPëCŸ");

```

```

42      System.out.println("8 — PñC,PjPµPSP° CÍPSC¼C,PëC¼");
43      System.out.println("9 — PñC,PjPµPSP° PìPµCṪPµPIPsPrP°");
44      System.out.println("10 — P¼CṪPsPjPsC,PëP° PICṪPµPjPµPSPë");
45      System.out.println("exit — program finish");
46      str = in.nextLine();
47      switch (str)
48      {
49          case "1":
50              PercentOfDeposit percent = null;
51              for (int j = 0; j < 2; j++)
52              {
53                  System.out.println("P'PIPµPrPëC,Pµ CÍCřPjPjCř Pë
PṪCṪPsCṪPµPSC," );
54                  percent.addPercentAndSum(in.nextInt(), in.
nextDouble());
55              }
56              System.out.println("P'PIPµPrPëC,Pµ PìP°CṪP°PjPµC,CṪC<
P±P°PSPëP°");
57
58              bank = new Bank(in.nextLine(), in.nextDouble(), in
.nextDouble(), in.nextDouble(), in.nextDouble(), percent);
59              central.registerBank(bank);
60              break;
61          case "2":
62              bank.addClientInBank(client);
63              break;
64          case "3":
65              switch (in.nextLine())
66              {
67                  case "1":
68                      System.out.println("P'PIPµPrPëC,Pµ CÍCřPjPjCř
Pë CÍCṪPsPe");
69                      central.creationAccount(bank, in.nextInt()
, client, in.nextInt(), "Credit");
70                      break;
71                  case "2":
72                      System.out.println("P'PIPµPrPëC,Pµ CÍCřPjPjCř
Pë CÍCṪPsPe");
73                      central.creationAccount(bank, in.nextInt()
, client, in.nextInt(), "Debit");
74                      break;
75                  case "3":
76                      System.out.println("P'PIPµPrPëC,Pµ CÍCřPjPjCř
Pë CÍCṪPsPe");
77                      central.creationAccount(bank, in.nextInt()
, client, in.nextInt(), "Deposit");
78                      break;
79                  default:
80                      throw new BanksException("Invalid option")
;

```

```

81         }
82
83         break;
84     case "4":
85         System.out.println("P'PIPμPrPëC,Pμ C'CrPjPjCr Pë
PSPsPjPμCTb C'CrPμC,P°");
86         central.withdrawalMoney(bank, in.nextInt(), in.
nextInt(), client);
87         break;
88     case "5":
89         System.out.println("P'PIPμPrPëC,Pμ C'CrPjPjCr Pë
PSPsPjPμCTb C'CrPμC,P°");
90         central.refillMoneyOn(bank, in.nextInt(), in.
nextInt());
91         break;
92     case "6":
93         System.out.println("P'PIPμPrPëC,Pμ C'CrPjPjCr,
PSPsPjPμCTb C'CrPμC,P° PsC,PiCTbP°PIPëC,PμP»Cμ Pë PSPsPjPμCTb C'CrPμC,P°
PìPsP»CrC†P°C,PμP»Cμ");
94         central.transferMoneyOnBalance(bank, in.nextInt(),
in.nextInt(), bank, in.nextInt(), client);
95         break;
96     case "7":
97         System.out.println("P'PIPμPrPëC,Pμ C'CrPjPjCr Pë
PSPsPjPμCTb C'CrPμC,P° ");
98         central.cancelRefill(bank, in.nextInt(), in.
nextInt());
99         break;
100    case "8":
101        System.out.println("P'PIPμPrPëC,Pμ C'CrPjPjCr Pë
PSPsPjPμCTb C'CrPμC,P° ");
102        central.cancelWithdrawal(bank, in.nextInt(), in.
nextInt());
103        break;
104    case "9":
105        System.out.println("P'PIPμPrPëC,Pμ C'CrPjPjCr,
PSPsPjPμCTb C'CrPμC,P° PsC,PiCTbP°PIPëC,PμP»Cμ Pë PSPsPjPμCTb C'CrPμC,P°
PìPsP»CrC†P°C,PμP»Cμ");
106        central.cancelTransfer(bank, in.nextInt(), in.
nextInt(), bank, in.nextInt());
107        break;
108    case "10":
109        System.out.println("P'PIPμPrPëC,Pμ C'CrPjPjCr Pë
PSPsPjPμCTb C'CrPμC,P° ");
110        central.checkTime( in.nextInt(), bank.getAccountld
( in.nextInt()));
111        break;
112
113    case "exit":
114        System.out.println("finish");

```

```
115 break ;  
116 }  
117 }  
118 }  
119 }
```

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
25 client, int time, String option) throws BanksException {
26         if (option == null) throw new BanksException("Invalid option");
27 ;
28         if (client == null) throw new BanksException("Invalid client");
29 ;
30         if (bank == null) throw new BanksException("Invalid Bank");
31         if (banks.stream().anyMatch(b -> b == bank)) return
32 creationAccountForClient(bank, client, balance, time, option);
33         else throw new BanksException("Invalid bank");
34     }
35
36     public void checkTime(int time, IAccount account) throws
37 BanksException {
38         if (account == null) throw new BanksException("Invalid account
39 ");
40         account.benefitPay(time);
41     }
42
43     public IAccount creationAccountForClient(Bank bank, Client client,
44 int balance, int time, String option) throws BanksException {
45         switch (option)
46         {
47             case "Credit":
48
49                 Credit credit = new Credit(countOfId++, bank.
```

```

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

```

```
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: Application.java

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

Листинг 1.14: CatController.java

```
1 package com.kisssusha.controller;
2
3 import com.kisssusha.DAO.dto.CatsDto;
4 import com.kisssusha.DAO.dto.FriendshipDto;
5 import com.kisssusha.service.KotikiService;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.web.bind.annotation.*;
8
9 import java.util.List;
10
11 @RestController
12 @RequestMapping("/cat")
13 public class CatController {
14     @Autowired
15     KotikiService service;
16
17     @GetMapping("/find-all-cats")
18     public List<CatsDto> findAllCats() {
19         return service.getCats();
20     }
21
22     @PostMapping("/add-cat")
23     public boolean addCat(@RequestBody CatsDto catsDto) {
24         return service.addCat(catsDto);
25     }
26
27     @DeleteMapping("/delete-cat/{id}")
28     public boolean deleteCat(@PathVariable("id") Long id) {
29         return service.deleteCat(id);
30     }
31
32     @DeleteMapping("/break-friendship")
33     public boolean breakFriendship(@RequestBody FriendshipDto
34     friendshipDto) {
35         return service.breakFriendship(friendshipDto.getIdCat(),
36     friendshipDto.getIdFriend());
37     }
38     @PostMapping("/make-friendship")
39     public boolean makeFriendship(@RequestBody FriendshipDto
40     friendshipDto){
41         return service.makeFriendship(friendshipDto);
42     }
43 }
```

Листинг 1.15: OwnerController.java

```
1 package com.kisssusha.controller;
2
3 import com.kisssusha.DAO.dto.OwnersDto;
4 import com.kisssusha.DAO.dto.ShelterDto;
5 import com.kisssusha.service.KotikiService;
6 import com.kisssusha.service.tools.KotikiException;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.web.bind.annotation.*;
9
10 import java.util.List;
11
12 @RestController
13 @RequestMapping("/owner")
14 public class OwnerController {
15     @Autowired
16     KotikiService service;
17
18     @GetMapping("/findAllOwner")
19     public List<OwnersDto> findAllOwner() {
20         return service.getOwners();
21     }
22
23     @PostMapping("/addOwner")
24     public boolean addOwner(@RequestBody OwnersDto ownersDto) {
25         return service.addOwner(ownersDto);
26     }
27
28     @DeleteMapping("/deleteOwner/{id}")
29     public boolean deleteOwner(@PathVariable("id") Long id) {
30         return service.deleteOwner(id);
31     }
32
33     @DeleteMapping("/break-shelter")
34     public boolean breakShelter(@RequestBody ShelterDto shelterDto)
35     throws KotikiException {
36         return service.breakShelter(shelterDto.getIdCat(), shelterDto.getIdOwner());
37     }
38
39     @PostMapping("/make-shelter")
40     public boolean makeShelter(@RequestBody ShelterDto shelterDto) {
41         return service.makeShelter(shelterDto);
42     }
43 }
```

Листинг 1.16: CatsDto.java

```
1 package com.kisssusha.DAO.dto;  
2  
3 import com.kisssusha.DAO.enums.MyColors;  
4 import com.kisssusha.DAO.models.Cats;  
5  
6 import java.sql.Timestamp;  
7 import java.util.Objects;  
8  
9 public class CatsDto {  
10     private Long id;  
11     private String name;  
12     private Timestamp birth;  
13     private String breed;  
14     private MyColors color;  
15  
16     public CatsDto() {  
17     }  
18  
19     public Long getId() {  
20         return id;  
21     }  
22  
23     public void setId(Long id) {  
24         this.id = id;  
25     }  
26  
27     public String getName() {  
28         return name;  
29     }  
30  
31     public void setName(String name) {  
32         this.name = name;  
33     }  
34  
35     public Timestamp getBirth() {  
36         return birth;  
37     }  
38  
39     public void setBirth(Timestamp birth) {  
40         this.birth = birth;  
41     }  
42  
43     public String getBreed() {  
44         return breed;  
45     }  
46  
47     public void setBreed(String breed) {  
48         this.breed = breed;  
49     }  
}
```

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

Листинг 1.17: FriendshipDto.java

```
1 package com.kisssusha.DAO.dto;  
2  
3 import com.kisssusha.DAO.models.Friendship;  
4  
5 import java.util.Objects;  
6  
7 public class FriendshipDto {  
8     private Long id;  
9     private Long idFriend;  
10    private Long idCat;  
11  
12    public FriendshipDto() {  
13    }  
14  
15    public Long getId() {  
16        return id;  
17    }  
18  
19    public void setId(Long id) {  
20        this.id = id;  
21    }  
22  
23    public Long getIdFriend() {  
24        return idFriend;  
25    }  
26  
27    public void setIdFriend(Long idFriend) {  
28        this.idFriend = idFriend;  
29    }  
30  
31    public Long getIdCat() {  
32        return idCat;  
33    }  
34  
35    public void setIdCat(Long idCat) {  
36        this.idCat = idCat;  
37    }  
38  
39    public FriendshipDto(Friendship friends) {  
40        this.id = friends.getId();  
41        this.idFriend = friends.getIdFriend();  
42        this.idCat = friends.getIdCat();  
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        FriendshipDto that = (FriendshipDto) o;
```

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

Листинг 1.18: OwnersDto.java

```
1 package com.kisssusha.DAO.dto;
2
3 import com.kisssusha.DAO.models.Owners;
4
5 import java.sql.Timestamp;
6 import java.util.Objects;
7
8 public class OwnersDto {
9     private Long id;
10    private String name;
11    private Timestamp date;
12
13    public OwnersDto(){
14    }
15
16    public Long getId() {
17        return id;
18    }
19
20    public void setId(Long id) {
21        this.id = id;
22    }
23
24    public String getName() {
25        return name;
26    }
27
28    public void setName(String name) {
29        this.name = name;
30    }
31
32    public Timestamp getDate() {
33        return date;
34    }
35
36    public void setDate(Timestamp date) {
37        this.date = date;
38    }
39
40    public OwnersDto(Owners owners){
41        this.id = owners.getId();
42        this.date = owners.getDate();
43        this.name = owners.getName();
44    }
45
46    @Override
47    public boolean equals(Object o) {
48        if (this == o) return true;
49        if (o == null || getClass() != o.getClass()) return false;
```



```
50         OwnersDto ownersDto = (OwnersDto) o;  
51         return id.equals(ownersDto.id) && name.equals(ownersDto.name)  
52         && date.equals(ownersDto.date);  
53     }  
54     @Override  
55     public int hashCode() {  
56         return Objects.hash(id, name, date);  
57     }  
58 }
```

Листинг 1.19: ShelterDto.java

```
1 package com.kisssusha.DAO.dto;
2
3 import com.kisssusha.DAO.models.Shelter;
4
5 import java.util.Objects;
6
7 public class ShelterDto {
8     private Long id;
9     private Long idCat;
10    private Long idOwner;
11    public ShelterDto(){
12    }
13
14    public Long getId() {
15        return id;
16    }
17
18    public void setId(Long id) {
19        this.id = id;
20    }
21
22    public Long getIdCat() {
23        return idCat;
24    }
25
26    public void setIdCat(Long idCat) {
27        this.idCat = idCat;
28    }
29
30    public Long getIdOwner() {
31        return idOwner;
32    }
33
34    public void setIdOwner(Long idOwner) {
35        this.idOwner = idOwner;
36    }
37
38    public ShelterDto(Shelter shelter){
39        this.id = shelter.getId();
40        this.idCat = shelter.getIdCat();
41        this.idOwner = shelter.getIdOwner();
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        ShelterDto that = (ShelterDto) o;
49        return Objects.equals(id, that.id) && Objects.equals(idCat,
```

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

Листинг 1.20: MyColors.java

```
1 package com.kisssusha.DAO.enums;  
2  
3 public enum MyColors {  
4     White ,  
5     Brown ,  
6     Black ,  
7     Orange  
8 }
```

Листинг 1.21: CatsDao.java

```
1 package com.kisssusha.DAO.implemetations;  
2  
3 import com.kisssusha.DAO.models.Cats;  
4 import org.springframework.data.jpa.repository.JpaRepository;  
5 import org.springframework.stereotype.Repository;  
6  
7 @Repository  
8 public interface CatsDao extends JpaRepository<Cats, Long> {  
9 }
```

Листинг 1.22: FriendshipDao.java

```
1 package com.kisssusha.DAO.implemetations;
2
3 import com.kisssusha.DAO.models.Friendship;
4
5 import org.springframework.data.jpa.repository.JpaRepository;
6 import org.springframework.stereotype.Repository;
7
8 @Repository
9 public interface FriendshipDao extends JpaRepository<Friendship, Long>
10 {
11     void deleteAllByIdCatOrIdFriend(Long idCat, Long idFriend);
12     void deleteAllByIdCatAndIdFriend(Long idCat, Long idFriend);
13 }
```

Листинг 1.23: OwnersDao.java

```
1 package com.kisssusha.DAO.implemetations;  
2  
3 import com.kisssusha.DAO.models.Owners;  
4 import org.springframework.data.jpa.repository.JpaRepository;  
5 import org.springframework.stereotype.Repository;  
6  
7 @Repository  
8 public interface OwnersDao extends JpaRepository<Owners, Long> {  
9  
10 }
```

Листинг 1.24: ShelterDao.java

```
1 package com.kisssusha.DAO.implemetations;
2
3 import com.kisssusha.DAO.models.Shelter;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 @Repository
8 public interface ShelterDao extends JpaRepository<Shelter, Long> {
9     void deleteAllByIdCat(Long catId);
10    void deleteAllByIdOwner(Long ownerId);
11    void deleteAllByIdOwnerAndIdCat(Long ownerId, Long catId);
12 }
```


Листинг 1.25: Cats.java

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

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

Листинг 1.26: Friendship.java

```
1 package com.kisssusha.DAO.models;
2
3 import com.kisssusha.DAO.dto.FriendshipDto;
4
5 import javax.persistence.*;
6 import java.util.Objects;
7
8 @Entity
9 @Table(name = "friend_for_cat", schema = "public", catalog = "postgres
10 ")
11 public class Friendship {
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     @Id
14     @Column(name = "id", nullable = false)
15     private Long id;
16     @Basic
17     @Column(name = "first_cat_id", nullable = false)
18     private Long idFriend;
19     @Basic
20     @Column(name = "second_cat_id", nullable = false)
21     private Long idCat;
22
23     public Friendship() {
24     }
25
26     public Long getId() {
27         return id;
28     }
29
30     public void setId(Long id) {
31         this.id = id;
32     }
33
34     public Long getIdFriend() {
35         return idFriend;
36     }
37
38     public void setIdFriend(Long idFriend) {
39         this.idFriend = idFriend;
40     }
41
42     public Long getIdCat() {
43         return idCat;
44     }
45
46     public void setIdCat(Long idCat) {
47         this.idCat = idCat;
48     }
49 }
```

```
49     public Friendship(FriendshipDto friends) {
50         this.idCat = friends.getIdCat();
51         this.idFriend = friends.getIdFriend();
52     }
53
54     @Override
55     public boolean equals(Object o) {
56         if (this == o) return true;
57         if (o == null || getClass() != o.getClass()) return false;
58         Friendship that = (Friendship) o;
59         return Objects.equals(id, that.id) && Objects.equals(idFriend,
60             that.idFriend)
61             && Objects.equals(idCat, that.idCat);
62     }
63
64     @Override
65     public int hashCode() {
66         return Objects.hash(id, idFriend, idCat);
67     }
```

Листинг 1.27: Owners.java

```
1 package com.kisssusha.DAO.models;
2
3 import com.kisssusha.DAO.dto.OwnersDto;
4
5 import javax.persistence.*;
6 import java.sql.Timestamp;
7 import java.util.Objects;
8
9 @Entity
10 @Table(name = "owners", schema = "public", catalog = "postgres")
11 public class Owners {
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 = "date", nullable = true)
21     private Timestamp date;
22
23     public Owners() {
24     }
25
26     public Long getId() {
27         return id;
28     }
29
30     public void setId(Long id) {
31         this.id = id;
32     }
33
34     public String getName() {
35         return name;
36     }
37
38     public void setName(String name) {
39         this.name = name;
40     }
41
42     public Timestamp getDate() {
43         return date;
44     }
45
46     public void setDate(Timestamp date) {
47         this.date = date;
48     }
49 }
```

```
50     public Owners(OwnersDto own) {
51         this.date = own.getDate();
52         this.name = own.getName();
53     }
54
55     @Override
56     public boolean equals(Object o) {
57         if (this == o) return true;
58         if (o == null || getClass() != o.getClass()) return false;
59         Owners that = (Owners) o;
60         return Objects.equals(id, that.id) && Objects.equals(name,
that.name)
61             && Objects.equals(date, that.date);
62     }
63
64     @Override
65     public int hashCode() {
66         return Objects.hash(id, name, date);
67     }
68 }
```

Листинг 1.28: Shelter.java

```
1 package com.kisssusha.DAO.models;
2
3 import com.kisssusha.DAO.dto.ShelterDto;
4
5 import javax.persistence.*;
6 import java.util.Objects;
7
8 @Entity
9 @Table(name = "shelter", schema = "public", catalog = "postgres")
10 public class Shelter {
11     @GeneratedValue(strategy = GenerationType.IDENTITY)
12     @Id
13     @Column(name = "id", nullable = false)
14     private Long id;
15     @Basic
16     @Column(name = "cat_id", nullable = false)
17     private Long idCat;
18     @Basic
19     @Column(name = "owner_id", nullable = false)
20     private Long idOwner;
21
22     public Shelter() {
23     }
24
25     public Long getId() {
26         return id;
27     }
28
29     public void setId(Long id) {
30         this.id = id;
31     }
32
33     public Long getIdCat() {
34         return idCat;
35     }
36
37     public void setIdCat(Long idCat) {
38         this.idCat = idCat;
39     }
40
41     public Long getIdOwner() {
42         return idOwner;
43     }
44
45     public void setIdOwner(Long idOwner) {
46         this.idOwner = idOwner;
47     }
48
49     public Shelter(ShelterDto shelter) {
```

```
50         this.idCat = shelter.getIdCat();
51         this.idOwner = shelter.getIdOwner();
52     }
53
54
55     @Override
56     public boolean equals(Object o) {
57         if (this == o) return true;
58         if (o == null || getClass() != o.getClass()) return false;
59         Shelter that = (Shelter) o;
60         return Objects.equals(id, that.id) && Objects.equals(idCat,
61 that.idCat)
62             && Objects.equals(idOwner, that.idOwner);
63     }
64
65     @Override
66     public int hashCode() {
67         return Objects.hash(id, idCat, idOwner);
68     }
69 }
```


Листинг 1.29: DaoException.java

```
1 package com.kisssusha.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 1.30: KotikiService.java

```
1 package com.kisssusha.service;
2
3 import com.kisssusha.DAO.dto.CatsDto;
4 import com.kisssusha.DAO.dto.FriendshipDto;
5 import com.kisssusha.DAO.dto.OwnersDto;
6 import com.kisssusha.DAO.dto.ShelterDto;
7 import com.kisssusha.DAO.implemetations.CatsDao;
8 import com.kisssusha.DAO.implemetations.FriendshipDao;
9 import com.kisssusha.DAO.implemetations.OwnersDao;
10 import com.kisssusha.DAO.implemetations.ShelterDao;
11 import com.kisssusha.DAO.models.Cats;
12 import com.kisssusha.DAO.models.Friendship;
13 import com.kisssusha.DAO.models.Owners;
14 import com.kisssusha.DAO.models.Shelter;
15 import com.kisssusha.service.tools.KotikiException;
16 import org.springframework.beans.factory.annotation.Autowired;
17 import org.springframework.stereotype.Service;
18 import org.springframework.transaction.annotation.Transactional;
19
20 import java.util.List;
21 import java.util.stream.Collectors;
22
23 @Service
24 public class KotikiService {
25     @Autowired
26     CatsDao catDAO;
27     @Autowired
28     OwnersDao ownerDAO;
29     @Autowired
30     FriendshipDao friendshipDao;
31     @Autowired
32     ShelterDao shelterDao;
33
34     public List<CatsDto> getCats(){
35         return catDAO.findAll().stream().map(CatsDto::new).collect(
36             Collectors.toList());
37     }
38
39     public boolean addCat(CatsDto catsDto){
40         catDAO.save(new Cats(catsDto));
41         return true;
42     }
43
44     public boolean makeShelter(ShelterDto shelterDto) {
45         shelterDao.save(new Shelter(shelterDto));
46         return true;
47     }
48
49     public boolean makeFriendship(FriendshipDto friendshipDto) {
50         friendshipDao.save(new Friendship(friendshipDto));
51     }
52 }
```

```
49         return true;
50     }
51     public boolean breakFriendship(Long firstCatId , Long secondCatId)
52     {
53         friendshipDao.deleteAllByIdCatAndIdFriend( firstCatId ,
54         secondCatId);
55         return true;
56     }
57     public boolean breakShelter(Long catId , Long ownerId) throws
58     KotikiException {
59         shelterDao.deleteAllByIdOwnerAndIdCat(ownerId , catId);
60         return true;
61     }
62
63     @Transactional
64     public boolean deleteCat(Long idCat){
65         shelterDao.deleteAllByIdCat(idCat);
66         friendshipDao.deleteAllByIdCatOrIdFriend(idCat , idCat);
67         catDAO.deleteById(idCat);
68         return true;
69     }
70
71     public List<OwnersDto> getOwners(){
72         return ownerDAO.findAll().stream().map(OwnersDto::new).collect
73         (Collectors.toList());
74     }
75
76     public boolean addOwner(OwnersDto ownersDto){
77         ownerDAO.save(new Owners(ownersDto));
78         return true;
79     }
80
81     @Transactional
82     public boolean deleteOwner(Long idOwner){
83         shelterDao.deleteAllByIdOwner(idOwner);
84         ownerDAO.deleteById(idOwner);
85         return true;
86     }
87 }
```

Листинг 1.31: KotikiException.java

```
1 package com.kisssusha.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 1.32: CatControllerTest.java

```
1 package com.kisssusha.controller;
2
3 import com.kisssusha.DAO.implemetations.CatsDao;
4 import com.kisssusha.DAO.implemetations.OwnersDao;
5 import com.kisssusha.service.KotikiService;
6 import org.junit.jupiter.api.Test;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.boot.test.autoconfigure.web.servlet.
    WebMvcTest;
9 import org.springframework.boot.test.mock.mockito.MockBean;
10 import org.springframework.test.web.servlet.MockMvc;
11
12 import static org.springframework.test.web.servlet.request.
    MockMvcRequestBuilders.get;
13 import static org.springframework.test.web.servlet.result.
    MockMvcResultMatchers.status;
14
15 @WebMvcTest(CatController.class)
16 public class CatControllerTest {
17     @MockBean
18     CatsDao catDAO;
19
20     @MockBean
21     OwnersDao ownerDAO;
22
23     @MockBean
24     KotikiService service;
25
26     @Autowired
27     private MockMvc mockMvc;
28
29     @Test
30     void checkCatsColor() throws Exception {
31         String url = "http://localhost:8080/cats/findAllOneColorCats/
    Pink";
32         mockMvc.perform(get(url)).andExpect(status().is4xxClientError
    ());
33     }
34 }
```