

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:

Жарков Григорий Алексеевич
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: Console.java

```

1 package controller;
2
3 import controller.tools.ControllerException;
4 import dao.colors.Color;
5 import dao.entities.Cat;
6 import dao.entities.Owner;
7 import service.Service;
8 import service.tools.ServiceException;
9
10 import java.sql.Timestamp;
11 import java.util.List;
12 import java.util.Scanner;
13
14 public class Console {
15     private Service service;
16
17     public Console(Service service) {
18         this.service = service;
19     }
20
21     public void work() throws ControllerException {
22         while (true) {
23             System.out.println("""
24                 Enter what you want to do:\s
25                 1) Add new cat\s
26                 2) Add new owner\s
27                 3) Add cat to owner\s
28                 4) Remove cat from owner\s
29                 5) Make friends\s
30                 6) Break friendship\s
31                 7) Remove cat\s
32                 8) Remove owner\s
33                 0) Exit""");
34
35             Scanner choice = new Scanner(System.in);
36             String answer = choice.nextLine();
37
38             switch (answer) {
39                 case "1" -> {
40                     System.out.println("Enter cat's name:");
41                     String name = choice.nextLine();
42
43                     System.out.println("Enter cat's birthdate (yyyy-mm
44 -dd):");
45
46                     String birthdate = choice.nextLine();
47
48                     System.out.println("Enter cat's species:");
49                     String species = choice.nextLine();
50
51                     service.addCat(name, birthdate, species);
52                 }
53             }
54         }
55     }
56 }

```

```
49         System.out.println("Enter cat's color (Black ,
White , Gray or Orange):");
50         String color = choice.nextLine();
51
52         birthdate = birthdate.concat(" 00:00:00");
53
54         try {
55             service.addCat(name, Timestamp.valueOf(
birthdate), species, Color.valueOf(color));
56         } catch (ServiceException e) {
57             throw new ControllerException("Problems with
adding cat to service" + e.getMessage());
58         }
59     }
60
61     case "2" -> {
62         System.out.println("Enter owner's name:");
63         String name = choice.nextLine();
64
65         System.out.println("Enter owner's birthdate (yyyy-
mm-dd):");
66         String birthdate = choice.nextLine();
67
68         birthdate = birthdate.concat(" 00:00:00");
69
70         try {
71             service.addOwner(name, Timestamp.valueOf(
birthdate));
72         } catch (ServiceException e) {
73             throw new ControllerException("Problems with
adding owner to service" + e.getMessage());
74         }
75     }
76
77     case "3" -> {
78         List<Cat> cats = service.getAllCats();
79         List<Owner> owners = service.getAllOwners();
80
81         System.out.println("Choose one cat to add to owner
:");
82         for (Cat cat: cats) {
83             System.out.println(cat);
84         }
85         String catNumber = choice.nextLine();
86
87         if (Integer.parseInt(catNumber) >= cats.size()) {
88             throw new ArrayIndexOutOfBoundsException();
89         }
90     }
```

```

91         System.out.println("Choose one owner to add cat to
:");
92         for (Owner owner: owners) {
93             System.out.println(owner);
94         }
95         String ownerNumber = choice.nextLine();
96
97         if (Integer.parseInt(ownerNumber) >= owners.size()
) {
98             throw new ArrayIndexOutOfBoundsException();
99         }
100
101         try {
102             service.addCatToOwner(cats.get(Integer.
parseInt(catNumber)), owners.get(Integer.parseInt(ownerNumber)));
103         } catch (ServiceException e) {
104             throw new ControllerException("Problems with
adding cat to service" + e.getMessage());
105         }
106     }
107
108     case "4" -> {
109         List<Cat> cats = service.getAllCats();
110         List<Owner> owners = service.getAllOwners();
111
112         System.out.println("Choose one cat to be removed
from owner:");
113         for (Cat cat: cats) {
114             System.out.println(cat);
115         }
116         String catNumber = choice.nextLine();
117
118         if (Integer.parseInt(catNumber) >= cats.size()) {
119             throw new ArrayIndexOutOfBoundsException();
120         }
121
122         System.out.println("Choose one owner whose cat
will be removed:");
123         for (Owner owner: owners) {
124             System.out.println(owner);
125         }
126         String ownerNumber = choice.nextLine();
127
128         if (Integer.parseInt(ownerNumber) >= owners.size()
) {
129             throw new ArrayIndexOutOfBoundsException();
130         }
131
132         try {

```

```
133         service.removeCatFromOwner(cats.get(Integer.parseInt(catNumber)), owners.get(Integer.parseInt(ownerNumber)));
134     } catch (ServiceException e) {
135         throw new ControllerException("Problems with removing cat from owner" + e.getMessage());
136     }
137 }
138
139     case "5" -> {
140         List<Cat> cats = service.getAllCats();
141
142         System.out.println("Choose first cat:");
143         for (Cat cat: cats) {
144             System.out.println(cat);
145         }
146         String firstCatNumber = choice.nextLine();
147
148         if (Integer.parseInt(firstCatNumber) >= cats.size
149     )) {
150             throw new ArrayIndexOutOfBoundsException();
151         }
152
153         System.out.println("Choose second cat:");
154         for (Cat cat: cats) {
155             System.out.println(cat);
156         }
157         String secondCatNumber = choice.nextLine();
158
159         if (Integer.parseInt(secondCatNumber) >= cats.size
160     )) {
161             throw new ArrayIndexOutOfBoundsException();
162         }
163
164         try {
165             service.makeFriends(cats.get(Integer.parseInt(firstCatNumber)), cats.get(Integer.parseInt(secondCatNumber)));
166         } catch (ServiceException e) {
167             throw new ControllerException("Problems with making friendship" + e.getMessage());
168         }
169     }
170
171     case "6" -> {
172         List<Cat> cats = service.getAllCats();
173
174         System.out.println("Choose first cat:");
175         for (Cat cat: cats) {
176             System.out.println(cat);
```

```
177         if (Integer.parseInt(firstCatNumber) >= cats.size
178     )) {
179             throw new ArrayIndexOutOfBoundsException();
180         }
181
182         System.out.println("Choose second cat:");
183         for (Cat cat: cats) {
184             System.out.println(cat);
185         }
186         String secondCatNumber = choice.nextLine();
187
188     )) {
189         if (Integer.parseInt(secondCatNumber) >= cats.size
190     )) {
191             throw new ArrayIndexOutOfBoundsException();
192         }
193
194         try {
195             service.breakFriendship(cats.get(Integer.
196     parseInt(firstCatNumber)), cats.get(Integer.parseInt(
197     secondCatNumber)));
198         } catch (ServiceException e) {
199             throw new ControllerException("Problems with
200     breaking friendship" + e.getMessage());
201         }
202     }
203
204     case "7" -> {
205         List<Cat> cats = service.getAllCats();
206
207         System.out.println("Choose cat to be removed:");
208         for (Cat cat: cats) {
209             System.out.println(cat);
210         }
211         String catNumber = choice.nextLine();
212
213         if (Integer.parseInt(catNumber) >= cats.size()) {
214             throw new ArrayIndexOutOfBoundsException();
215         }
216
217         try {
218             service.removeCat(cats.get(Integer.parseInt(
219     catNumber)));
220         } catch (ServiceException e) {
221             throw new ControllerException("Problems with
222     removing cat from service" + e.getMessage());
223         }
224     }
225
226     case "8" -> {
```

```
220         List<Owner> owners = service.getAllOwners();
221
222         System.out.println("Choose owner to be removed:");
223         for (Owner owner: owners) {
224             System.out.println(owner);
225         }
226         String ownerNumber = choice.nextLine();
227
228         if (Integer.parseInt(ownerNumber) >= owners.size()
229 ) {
230             throw new ArrayIndexOutOfBoundsException();
231         }
232         try {
233             service.removeOwner(owners.get(Integer.
234             parseInt(ownerNumber)));
235         } catch (ServiceException e) {
236             throw new ControllerException("Problems with
237             removing owner from service" + e.getMessage());
238         }
239         case "0" -> System.exit(0);
240
241         default -> {
242             var e = new IllegalArgumentException();
243             throw new ControllerException("Invalid option
244             number!", e);
245         }
246     }
247 }
248 }
```


Листинг 1.2: Main.java

```
1 package controller;  
2  
3 import controller.tools.ControllerException;  
4 import dao.implementations.CatDAO;  
5 import dao.implementations.CatsFriendsDAO;  
6 import dao.implementations.OwnerDAO;  
7 import dao.implementations.OwnersCatsDAO;  
8 import service.Service;  
9  
10 public class Main {  
11     public static void main(String[] args) throws ControllerException  
12     {  
13         Service service = new Service(new CatDAO(), new OwnerDAO(),  
14         new CatsFriendsDAO(), new OwnersCatsDAO());  
15         Console ui = new Console(service);  
16         ui.work();  
17     }  
18 }
```

Листинг 1.3: AccountType.java

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

Листинг 1.4: Credit.java

```
1 package account;
2
3 import tools.BankException;
4
5 import java.util.UUID;
6
7 public class Credit implements IAccount {
8     private double fee;
9     private double limit;
10
11     private double unverifiedLimit;
12     private boolean verified;
13
14     private double accumulatedFee;
15     private double balance;
16
17     private UUID id;
18
19     public Credit(double fee, double limit, boolean verified, double
unverifiedLimit) throws BankException {
20         if (fee <= 0) throw new BankException("Fee for credit account
must be positive!");
21         if (limit <= 0) throw new BankException("Limit for credit
account must be positive!");
22         if (unverifiedLimit <= 0) throw new BankException("Limit for
unverified account must be positive!");
23
24         this.fee = fee;
25         this.limit = limit;
26         this.unverifiedLimit = unverifiedLimit;
27         this.verified = verified;
28
29         id = UUID.randomUUID();
30
31         accumulatedFee = 0;
32         balance = 0;
33     }
34
35     @Override
36     public UUID getId() {
37         return id;
38     }
39
40     @Override
41     public double getBalance() {
42         return balance;
43     }
44
45     public void takeMoney(double amount) throws BankException {
```

```
46         if (amount <= 0) throw new BankException("Amount must be
positive!");
47         if (amount > balance + limit) throw new BankException("Amount
is too big!");
48         if (amount > unverifiedLimit && !verified)
49             throw new BankException("Amount is bigger than limit for
unverified account!");
50
51         balance -= amount;
52     }
53
54     public void addMoney(double amount) throws BankException {
55         if (amount <= 0) throw new BankException("Amount must be
positive!");
56         balance += amount;
57     }
58
59     public void calculateDailyPayment() {
60         if (balance < 0) {
61             accumulatedFee += fee;
62         }
63     }
64
65     public void getReward() throws BankException {
66         if (balance + limit < accumulatedFee) throw new BankException(
"Balance too low to take commission!");
67         balance -= accumulatedFee;
68         accumulatedFee = 0;
69     }
70
71     @Override
72     public boolean equals(Object obj) {
73         if (obj == this) return true;
74         if (obj == null || obj.getClass() != this.getClass()) return
false;
75
76         Credit other = (Credit) obj;
77
78         return other.getId() == this.getId();
79     }
80
81     @Override
82     public int hashCode() {
83         return id.hashCode();
84     }
85 }
```

Листинг 1.5: Debit.java

```
1 package account;
2
3 import tools.BankException;
4
5 import java.util.UUID;
6
7 public class Debit implements IAccount {
8     private double interest;
9
10    private double unverifiedLimit;
11    private boolean verified;
12
13    private double accumulatedAmount;
14    private double balance;
15
16    private UUID id;
17
18    public Debit(double interest, boolean verified, double
unverifiedLimit) throws BankException {
19        if (interest <= 0) throw new BankException("Interest for debit
account must be positive!");
20        if (unverifiedLimit <= 0) throw new BankException("Limit for
unverified account must be positive!");
21
22        this.interest = interest;
23        this.verified = verified;
24        this.unverifiedLimit = unverifiedLimit;
25
26        id = UUID.randomUUID();
27
28        accumulatedAmount = 0;
29        balance = 0;
30    }
31
32    @Override
33    public UUID getId() {
34        return id;
35    }
36
37    @Override
38    public double getBalance() {
39        return balance;
40    }
41
42    public void takeMoney(double amount) throws BankException {
43        if (amount <= 0) throw new BankException("Amount must be
positive!");
44        if (amount > balance) throw new BankException("Amount is too
big!");
```

```
45         if (amount > unverifiedLimit && !verified)
46             throw new BankException("Amount is bigger than limit for
unverified account!");
47
48         balance -= amount;
49     }
50
51     public void addMoney(double amount) throws BankException {
52         if (amount <= 0) throw new BankException("Amount must be
positive!");
53         balance += amount;
54     }
55
56     @Override
57     public void calculateDailyPayment() {
58         accumulatedAmount += balance * interest / 365;
59     }
60
61     @Override
62     public void getReward() {
63         balance += accumulatedAmount;
64         accumulatedAmount = 0;
65     }
66
67     @Override
68     public boolean equals(Object obj) {
69         if (obj == this) return true;
70         if (obj == null || obj.getClass() != this.getClass()) return
false;
71
72         Debit other = (Debit) obj;
73
74         return other.getId() == this.getId();
75     }
76
77     @Override
78     public int hashCode() {
79         return id.hashCode();
80     }
81 }
```

Листинг 1.6: Deposit.java

```
1 package account;
2
3 import tools.BankException;
4
5 import java.time.LocalDate;
6 import java.util.Map;
7 import java.util.UUID;
8
9 public class Deposit implements IAccount {
10     private double interest;
11
12     private double unverifiedLimit;
13     private boolean verified;
14
15     private LocalDate validUntil;
16
17     private double accumulatedAmount;
18     private double balance;
19
20     private UUID id;
21
22     public Deposit(double interest, Map<Double, Double>
interestConditions,
23         DepositDTO depositData, boolean verified, double
unverifiedLimit) throws BankException {
24         if (interest <= 0) throw new BankException("Interest for debit
account must be positive!");
25         if (unverifiedLimit <= 0) throw new BankException("Limit for
unverified account must be positive!");
26
27         for (Double percent : interestConditions.keySet()) {
28             if (percent <= 0) throw new BankException("Percent can not
be null!");
29         }
30
31         for (Double amount : interestConditions.values()) {
32             if (amount <= 0) throw new BankException("Amount can not
be null!");
33         }
34
35         if (depositData == null) {
36             var e = new IllegalArgumentException();
37             throw new BankException("Deposit data can not be null!", e
);
38         }
39
40         balance = depositData.getBalance();
41         validUntil = LocalDate.parse(depositData.getValidUntil());
42     }
```

```
43         if (balance <= 0) throw new BankException("Balance must be
positive!");
44         if (validUntil.isBefore(LocalDate.now())) throw new
BankException("Invalid date!");
45
46         this.interest = interest;
47         for (Map.Entry<Double, Double> condition : interestConditions.
entrySet()) {
48             if (balance <= condition.getKey()) this.interest =
condition.getValue();
49         }
50
51         this.verified = verified;
52         this.unverifiedLimit = unverifiedLimit;
53
54         id = UUID.randomUUID();
55         accumulatedAmount = 0;
56     }
57
58     @Override
59     public UUID getId() {
60         return id;
61     }
62
63     @Override
64     public double getBalance() {
65         return balance;
66     }
67
68     @Override
69     public void takeMoney(double amount) throws BankException {
70         if (amount <= 0) throw new BankException("Amount must be
positive!");
71         if (amount > balance) throw new BankException("Amount is too
big!");
72         if (amount > unverifiedLimit && !verified)
73             throw new BankException("Amount is bigger than limit for
unverified account!");
74
75         if (LocalDate.now().isBefore(validUntil)) throw new
BankException("It is impossible to take money now!");
76
77         balance -= amount;
78     }
79
80     @Override
81     public void addMoney(double amount) throws BankException {
82         if (amount <= 0) throw new BankException("Amount must be
positive!");
83         balance += amount;
```



```
84     }
85
86     @Override
87     public void calculateDailyPayment() {
88         accumulatedAmount += balance * interest / 365;
89     }
90
91     @Override
92     public void getReward() {
93         balance += accumulatedAmount;
94         accumulatedAmount = 0;
95     }
96
97     @Override
98     public boolean equals(Object obj) {
99         if (obj == this) return true;
100        if (obj == null || obj.getClass() != this.getClass()) return
false;
101
102        Deposit other = (Deposit) obj;
103
104        return other.getId() == this.getId();
105    }
106
107    @Override
108    public int hashCode() {
109        return id.hashCode();
110    }
111 }
```

Листинг 1.7: DepositConsole.java

```
1 package account;
2
3 import java.util.Scanner;
4
5 public class DepositConsole {
6     public DepositDTO collectDepositConditions() {
7         Scanner in = new Scanner(System.in);
8
9         System.out.println("Enter amount you want to deposit:");
10        double balance = in.nextDouble();
11
12        System.out.println("Enter date you want deposit will be valid
until (dd-mm-yyyy):");
13        String date = in.nextLine();
14
15        return new DepositDTO(balance, date);
16    }
17 }
```

Листинг 1.8: DepositDTO.java

```
1 package account;
2
3 public class DepositDTO {
4     private double balance;
5     private String validUntil;
6
7     public DepositDTO(double balance, String validUntil){
8         this.balance = balance;
9         this.validUntil = validUntil;
10    }
11
12    public double getBalance() {
13        return balance;
14    }
15
16    public String getValidUntil() {
17        return validUntil;
18    }
19 }
```

Листинг 1.9: IAccount.java

```
1 package account;
2
3 import tools.BankException;
4
5 import java.util.UUID;
6
7 public interface IAccount {
8
9     double getBalance();
10    UUID getId();
11
12    void takeMoney(double amount) throws BankException;
13    void addMoney(double amount) throws BankException;
14
15    void calculateDailyPayment();
16    void getReward() throws BankException;
17
18 }
```

Листинг 1.10: Bank.java

```
1 package bank;
2
3 import account.*;
4 import client.Client;
5 import client.ClientDTO;
6 import tools.BankException;
7 import tools.EventManager;
8 import tools.IEventListener;
9
10 import java.util.ArrayList;
11 import java.util.List;
12 import java.util.Map;
13
14 public class Bank implements IEventListener {
15
16     private CentralBank centralBank;
17     private List<Client> clients;
18
19     private String name;
20     private double debitInterest;
21     private double creditFee;
22     private double creditLimit;
23     private double unverifiedLimit;
24     private double depositDefaultInterest;
25     private Map<Double, Double> depositInterestConditions;
26
27     public EventManager events;
28
29     public Bank(CentralBank centralBank, BankDTO bankData) throws
BankException {
30         if (centralBank == null) {
31             var e = new IllegalArgumentException();
32             throw new BankException("Central bank can not be null!", e
);
33         }
34
35         if (bankData == null) {
36             var e = new IllegalArgumentException();
37             throw new BankException("Bank data can not be null!", e);
38         }
39
40         if (bankData.getName() == null) {
41             var e = new IllegalArgumentException();
42             throw new BankException("Bank data can not be null!", e);
43         }
44         if (bankData.getCreditFee() <= 0) throw new BankException("
Credit fee can not be negative!");
45         if (bankData.getCreditLimit() <= 0) throw new BankException("
Credit limit can not be negative!");
```

```
46         if (bankData.getDebitInterest() <= 0) throw new BankException(  
"Debit interest can not be negative!");  
47         if (bankData.getDepositDefaultInterest() <= 0) throw new  
BankException("Deposit default interest can not be negative!");  
48         if (bankData.getUnverifiedLimit() <= 0) throw new  
BankException("Unverified limit can not be negative!");  
49  
50         this.centralBank = centralBank;  
51  
52         name = bankData.getName();  
53  
54         debitInterest = bankData.getDebitInterest();  
55         creditFee = bankData.getCreditFee();  
56         creditLimit = bankData.getCreditLimit();  
57         unverifiedLimit = bankData.getUnverifiedLimit();  
58         depositDefaultInterest = bankData.getDepositDefaultInterest();  
59         depositInterestConditions = bankData.  
getDepositInterestConditions();  
60  
61         clients = new ArrayList<>();  
62  
63         events = new EventManager("unverified limit", "debit interest"  
, "credit fee", "credit limit");  
64     }  
65  
66     public String getName() {  
67         return name;  
68     }  
69  
70     public double getDebitInterest() {  
71         return debitInterest;  
72     }  
73  
74     public double getCreditFee() {  
75         return creditFee;  
76     }  
77  
78     public double getCreditLimit() {  
79         return creditLimit;  
80     }  
81  
82     public double getUnverifiedLimit() {  
83         return unverifiedLimit;  
84     }  
85  
86     public double getDepositDefaultInterest() {  
87         return depositDefaultInterest;  
88     }  
89  
90     public List<Client> getClients() {
```

```
91         return clients;
92     }
93
94     public Client registerClient(ClientDTO clientData) throws
BankException {
95         if (clientData == null) {
96             var e = new IllegalArgumentException();
97             throw new BankException("Client data can not be null!", e)
;
98         }
99
100         for (Client client: clients) {
101             if (client.getId() == clientData.getId()) {
102                 throw new BankException("Such client already exist!");
103             }
104         }
105
106         Client client = new Client(clientData, this);
107         clients.add(client);
108
109         return client;
110     }
111
112     public Client fillMissingData(Client client, ClientDTO clientData)
throws BankException {
113         if (client == null) {
114             var e = new IllegalArgumentException();
115             throw new BankException("Client can not be null!", e);
116         }
117
118         if (clientData == null) {
119             var e = new IllegalArgumentException();
120             throw new BankException("Client data can not be null!", e)
;
121         }
122
123         if (!clients.contains(client)) throw new BankException("
Unknown client!");
124
125         client.addMissingData(clientData);
126
127         return client;
128     }
129
130     public IAccount registerAccount(Client client, AccountType
accountType, DepositDTO depositData) throws BankException {
131         if (client == null) {
132             var e = new IllegalArgumentException();
133             throw new BankException("Client can not be null!", e);
134         }
```

```
135
136     IAccount account;
137     switch (accountType) {
138         case Credit -> account = new Credit(creditFee, creditLimit
139 , client.getVerified(), unverifiedLimit);
140         case Debit -> account = new Debit(debitInterest, client.
141 getVerified(), unverifiedLimit);
142         case Deposit -> account = new Deposit(
143 depositDefaultInterest, depositInterestConditions, depositData,
144 client.getVerified(), unverifiedLimit);
145         default -> {
146             var e = new IllegalArgumentException();
147             throw new BankException("Invalid account type!", e);
148         }
149     }
150     client.addAccount(account);
151     return account;
152 }
153
154 public void calculateDailyPayment() {
155     for (Client client: clients) {
156         for (IAccount account: client.getAccounts()) {
157             account.calculateDailyPayment();
158         }
159     }
160 }
161
162 public void payReward() throws BankException {
163     for (Client client: clients) {
164         for (IAccount account: client.getAccounts()) {
165             account.getReward();
166         }
167     }
168 }
169
170 public void changeUnverifiedLimit(double newLimit) throws
171 BankException {
172     if (newLimit <= 0) throw new BankException("Limit must be
173 positive!");
174     unverifiedLimit = newLimit;
175
176     events.notify("unverified limit");
177 }
178
179 public void changeDebitInterest(double newInterest) throws
180 BankException {
181     if (newInterest <= 0) throw new BankException("Interest must
182 be positive!");
183 }
```



```
177         debitInterest = newInterest;
178
179         events.notify("debit interest");
180     }
181
182     public void changeCreditFee(double newFee) throws BankException {
183         if (newFee <= 0) throw new BankException("Fee must be positive
184         !");
185         creditFee = newFee;
186
187         events.notify("credit fee");
188     }
189
190     public void changeCreditLimit(double newLimit) throws
191     BankException {
192         if (newLimit <= 0) throw new BankException("Limit must be
193         positive!");
194         creditFee = newLimit;
195
196         events.notify("credit limit");
197     }
198
199     @Override
200     public void update(String eventType) throws BankException {
201         switch (eventType) {
202             case "daily payment" -> calculateDailyPayment();
203             case "monthly payment" -> payReward();
204             default -> {
205                 var e = new IllegalArgumentException();
206                 throw new BankException("Invalid event!", e);
207             }
208         }
209     }
210
211     @Override
212     public boolean equals(Object obj) {
213         if (obj == this) return true;
214         if (obj == null || obj.getClass() != this.getClass()) return
215         false;
216
217         Bank other = (Bank) obj;
218
219         return other.getName().equals(this.getName());
220     }
221
222     @Override
223     public int hashCode() {
224         return name.hashCode();
225     }
226 }
```

Листинг 1.11: BankConsole.java

```
1 package bank;
2
3 import java.util.HashMap;
4 import java.util.Scanner;
5
6 public class BankConsole {
7     public BankDTO collectBankData() {
8         Scanner in = new Scanner(System.in);
9
10        System.out.println("Enter bank name:");
11        String name = in.nextLine();
12
13        System.out.println("Enter debit interest:");
14        double debitInterest = in.nextDouble();
15
16        System.out.println("Enter credit fee:");
17        double creditFee = in.nextDouble();
18
19        System.out.println("Enter credit limit:");
20        double creditLimit = in.nextDouble();
21
22        System.out.println("Enter limit for unverified clients:");
23        double unverifiedLimit = in.nextDouble();
24
25        System.out.println("Enter deposit default interest:");
26        double depositDefaultInterest = in.nextDouble();
27
28        System.out.println("Enter how many conditions will be");
29        int n = in.nextInt();
30
31        HashMap<Double, Double> conditions = new HashMap<>(n);
32
33        for (int i = 0; i < n; i++) {
34            System.out.println("Enter amount border:");
35            double amountBorder = in.nextDouble();
36
37            System.out.println("Enter interest for this border");
38            double interestBorder = in.nextDouble();
39
40            conditions.put(amountBorder, interestBorder);
41        }
42
43        return new BankDTO(name, debitInterest, creditFee, creditLimit
44            ,
45            unverifiedLimit, depositDefaultInterest, conditions);
46    }
```

Листинг 1.12: BankDTO.java

```
1 package bank;
2
3 import java.util.Map;
4
5 public class BankDTO {
6
7     private String name;
8     private double debitInterest;
9     private double creditFee;
10    private double creditLimit;
11    private double unverifiedLimit;
12    private double depositDefaultInterest;
13    private Map<Double, Double> depositInterestConditions;
14
15    public BankDTO(String name, double debitInterest, double creditFee
16    , double creditLimit,
17        double unverifiedLimit, double
18    depositDefaultInterest, Map<Double, Double>
19    depositInterestConditions) {
20        this.name = name;
21        this.debitInterest = debitInterest;
22        this.creditFee = creditFee;
23        this.creditLimit = creditLimit;
24        this.unverifiedLimit = unverifiedLimit;
25        this.depositDefaultInterest = depositDefaultInterest;
26        this.depositInterestConditions = depositInterestConditions;
27    }
28
29    public double getCreditFee() {
30        return creditFee;
31    }
32
33    public double getCreditLimit() {
34        return creditLimit;
35    }
36
37    public double getDebitInterest() {
38        return debitInterest;
39    }
40
41    public String getName() {
42        return name;
43    }
44
45    public double getDepositDefaultInterest() {
46        return depositDefaultInterest;
47    }
48
49    public double getUnverifiedLimit() {
```

```
47         return unverifiedLimit;
48     }
49
50     public Map<Double, Double> getDepositInterestConditions() {
51         return depositInterestConditions;
52     }
53 }
```

Листинг 1.13: CentralBank.java

```
1 package bank;
2
3 import account.IAccount;
4 import tools.BankException;
5 import tools.EventManager;
6
7 import java.time.Duration;
8 import java.time.LocalDate;
9 import java.util.ArrayList;
10 import java.util.List;
11 import java.util.Objects;
12
13 public class CentralBank {
14
15     private final List<Bank> banks;
16     private final List<Transaction> transactions;
17
18     public EventManager events;
19
20     public CentralBank() {
21         banks = new ArrayList<>();
22         transactions = new ArrayList<>();
23
24         events = new EventManager("daily payment", "monthly payment");
25     }
26
27     public List<Bank> getBanks() {
28         return banks;
29     }
30
31     public List<Transaction> getTransactions() {
32         return transactions;
33     }
34
35     public Bank addBank(BankDTO bankData) throws BankException {
36         if (bankData == null) {
37             var e = new IllegalArgumentException();
38             throw new BankException("Bank can not be null!", e);
39         }
40
41         for (Bank bank: banks) {
42             if (Objects.equals(bank.getName(), bankData.getName())) {
43                 throw new BankException("Bank with such name already
44 exist!");
45             }
46         }
47
48         Bank bank = new Bank(this, bankData);
49         banks.add(bank);
50     }
51 }
```

```
49     return bank;
50 }
51
52
53 public Transaction makeTransaction(IAccount from, double amount,
IAccount to) throws BankException {
54     if (from == null) {
55         var e = new IllegalArgumentException();
56         throw new BankException("Account can not be null!", e);
57     }
58
59     if (to == null) {
60         var e = new IllegalArgumentException();
61         throw new BankException("Account can not be null!", e);
62     }
63
64     if (amount <= 0) throw new BankException("Amount can not be
negative!");
65
66     from.takeMoney(amount);
67     to.addMoney(amount);
68
69     var transaction = new Transaction(from, amount, to);
70     transactions.add(transaction);
71
72     return transaction;
73 }
74
75 public void cancelTransaction(Transaction transaction) throws
BankException {
76     if (transaction == null) {
77         var e = new IllegalArgumentException();
78         throw new BankException("Transaction can not be null!", e)
;
79     }
80
81     transaction.getFrom().addMoney(transaction.getAmount());
82     transaction.getTo().takeMoney(transaction.getAmount());
83
84     transactions.remove(transaction);
85 }
86
87 public void calculateIncome(LocalDate from, LocalDate to) throws
BankException {
88     long daysBetween = Duration.between(from, to).toDays();
89     for (long i = 0; i < daysBetween; i++) {
90         events.notify("daily payment");
91         if (i % 30 == 0 && i > 0) events.notify("monthly payment")
;
92     }
```

93
94
95
96

}

}

Листинг 1.14: Transaction.java

```
1 package bank;
2
3 import account.IAccount;
4 import tools.BankException;
5
6 public class Transaction {
7     private final double amount;
8     private final IAccount from;
9     private final IAccount to;
10
11     public Transaction(IAccount from, double amount, IAccount to)
12     throws BankException {
13         if (from == null) {
14             var e = new IllegalArgumentException();
15             throw new BankException("Account can not be null!", e);
16         }
17
18         if (to == null) {
19             var e = new IllegalArgumentException();
20             throw new BankException("Account can not be null!", e);
21         }
22
23         if (amount <= 0) throw new BankException("Amount can not be
24         negative!");
25
26         this.amount = amount;
27         this.from = from;
28         this.to = to;
29     }
30
31     public double getAmount() {
32         return amount;
33     }
34
35     public IAccount getFrom() {
36         return from;
37     }
38
39     public IAccount getTo() {
40         return to;
41     }
42
43     @Override
44     public boolean equals(Object obj) {
45         if (obj == this) return true;
46         if (obj == null || obj.getClass() != this.getClass()) return
false;
47
48         Transaction other = (Transaction) obj;
```



```
47         return from.equals(other.from) && to.equals(other.to) &&
48         amount == other.amount;
49     }
50
51     @Override
52     public int hashCode() {
53         return from.hashCode() + to.hashCode() - (int) amount;
54     }
55 }
```

Листинг 1.15: Client.java

```
1 package client;
2
3 import account.IAccount;
4 import bank.Bank;
5 import tools.BankException;
6 import tools.IEventListener;
7
8 import java.util.ArrayList;
9 import java.util.List;
10 import java.util.UUID;
11
12 public class Client implements IEventListener {
13
14     private final String name;
15     private final String surname;
16
17     private String address;
18     private String passport;
19
20     private final UUID id;
21
22     private boolean verified;
23
24     private final List<IAccount> accounts;
25
26     private final Bank bank;
27
28     public Client(ClientDTO clientData, Bank bank) throws
BankException {
29
30         if (clientData == null){
31             var e = new IllegalArgumentException();
32             throw new BankException("Client data can not be null!", e)
33         ;
34         }
35
36         if (bank == null) {
37             var e = new IllegalArgumentException();
38             throw new BankException("Bank can not be null!", e);
39         }
40
41         this.bank = bank;
42
43         if (clientData.getName() == null) throw new
IllegalArgumentException();
44         if (clientData.getSurname() == null) throw new
IllegalArgumentException();
45
46         name = clientData.getName();
```

```
46         surname = clientData.getSurname();
47
48         if (clientData.getAddress() == null) throw new
IllegalArgumentException();
49         if (clientData.getPassport() == null) throw new
IllegalArgumentException();
50
51         verified = !clientData.getAddress().equals("LATER") && !
clientData.getPassport().equals("LATER");
52
53         address = clientData.getAddress();
54         passport = clientData.getPassport();
55
56         id = clientData.getId();
57         accounts = new ArrayList<>();
58     }
59
60     public String getName() {
61         return name;
62     }
63
64     public String getSurname() {
65         return surname;
66     }
67
68     public String getAddress() {
69         return address;
70     }
71
72     public String getPassport() {
73         return passport;
74     }
75
76     public UUID getId() {
77         return id;
78     }
79
80     public boolean getVerified() {
81         return verified;
82     }
83
84     public List<IAccount> getAccounts() {
85         return accounts;
86     }
87
88     public void addMissingData(ClientDTO clientData) throws
BankException {
89         if (clientData == null){
90             var e = new IllegalArgumentException();
91             throw new BankException("Client data can not be null!", e)
```

```
;
    }
    if (clientData.getAddress() == null) throw new
IllegalArgumentException();
    if (clientData.getPassport() == null) throw new
IllegalArgumentException();
    if (clientData.getAddress().equals("LATER")) throw new
BankException("Address must be valid!");
    if (clientData.getPassport().equals("LATER")) throw new
BankException("Passport must be valid!");
    address = clientData.getAddress();
    passport = clientData.getPassport();
    verified = true;
}

public void addAccount(IAccount account) throws BankException {
    if (account == null) {
        var e = new IllegalArgumentException();
        throw new BankException("Account can not be null!", e);
    }
    if (accounts.contains(account)) throw new BankException("This
client already has this account!");
    accounts.add(account);
}

public void displayEvent(String event) {
}

@Override
public void update(String eventType) throws BankException {
    displayEvent(eventType);
}

@Override
public boolean equals(Object obj) {
    if (obj == this) return true;
    if (obj == null || obj.getClass() != this.getClass()) return
false;
    Client other = (Client) obj;
    return other.getId() == this.getId();
}

@Override
```

```
135     public int hashCode() {  
136         return id.hashCode();  
137     }  
138 }
```

Листинг 1.16: ClientConsole.java

```
1 package client;
2
3 import tools.BankException;
4
5 import java.util.Scanner;
6
7 public class ClientConsole {
8     public ClientDTO collectPersonalData() {
9         Scanner in = new Scanner(System.in);
10
11         System.out.println("Enter your name:");
12         String name = in.nextLine();
13
14         System.out.println("Enter your surname:");
15         String surname = in.nextLine();
16
17         System.out.println("Enter your address:");
18         String address = in.nextLine();
19
20         System.out.println("Enter your passport:");
21         String passport = in.nextLine();
22
23         return new ClientDTO(name, surname, address, passport);
24     }
25
26     public ClientDTO addMissingData(ClientDTO clientData) throws
BankException {
27         if (clientData == null) {
28             var e = new IllegalArgumentException();
29             throw new BankException("Client data can not be null!", e)
;
30         }
31
32         String address = "LATER";
33         String passport = "LATER";
34         Scanner in = new Scanner(System.in);
35
36         if (clientData.getAddress().equals("LATER")) {
37             System.out.println("Enter your address^");
38             address = in.nextLine();
39         }
40
41         if (clientData.getPassport().equals("LATER")) {
42             System.out.println("Enter your passport");
43             passport = in.nextLine();
44         }
45
46         return new ClientDTO(clientData.getName(), clientData.
getSurname(), address, passport);
```

47
48

}
}

Листинг 1.17: ClientDTO.java

```
1 package client;
2
3 import java.util.UUID;
4
5 public class ClientDTO {
6
7     private final String name;
8     private final String surname;
9     private final String address;
10    private final String passport;
11    private final UUID id;
12
13    public ClientDTO(String name, String surname, String address,
14String passport) {
15        this.name = name;
16        this.surname = surname;
17        this.address = address;
18        this.passport = passport;
19
20        id = UUID.randomUUID();
21    }
22
23    public String getName() {
24        return name;
25    }
26
27    public String getSurname() {
28        return surname;
29    }
30
31    public String getAddress() {
32        return address;
33    }
34
35    public String getPassport() {
36        return passport;
37    }
38
39    public UUID getId() {
40        return id;
41    }
42 }
```


Листинг 1.18: BankException.java

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

Листинг 1.19: EventManager.java

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

Листинг 1.20: IEventListener.java

```
1 package tools;  
2  
3 public interface IEventListener {  
4     void update(String eventType) throws BankException;  
5 }
```

Листинг 1.21: ControllerException.java

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

Листинг 1.22: Color.java

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

Листинг 1.23: Cat.java

```
1 package dao.entities;
2
3 import dao.colors.Color;
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 Cat {
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     @Id
14     @Basic
15     @Column(name = "id")
16     private long id;
17     @Basic
18     @Column(name = "name")
19     private String name;
20     @Basic
21     @Column(name = "birthdate")
22     private Timestamp birthdate;
23     @Basic
24     @Column(name = "species")
25     private String species;
26     @Basic
27     @Column(name = "color")
28     @Enumerated(EnumType.STRING)
29     private Color color;
30
31     public long getId() {
32         return id;
33     }
34
35     public void setId(long id) {
36         this.id = id;
37     }
38
39     public String getName() {
40         return name;
41     }
42
43     public void setName(String name) {
44         this.name = name;
45     }
46
47     public Timestamp getBirthdate() {
48         return birthdate;
49     }
```

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

Листинг 1.24: CatsFriends.java

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



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

Листинг 1.25: Owner.java

```
1 package dao.entities;
2
3 import javax.persistence.*;
4 import java.sql.Timestamp;
5 import java.util.Objects;
6
7 @Entity
8 @Table(name = "owner", schema = "public", catalog = "postgres")
9 public class Owner {
10     @GeneratedValue(strategy = GenerationType.IDENTITY)
11     @Id
12     @Column(name = "id")
13     private long id;
14     @Basic
15     @Column(name = "name")
16     private String name;
17     @Basic
18     @Column(name = "birthdate")
19     private Timestamp birthdate;
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 getBirthdate() {
38         return birthdate;
39     }
40
41     public void setBirthdate(Timestamp birthdate) {
42         this.birthdate = birthdate;
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         Owner that = (Owner) o;
```

```
50         return id == that.id && Objects.equals(name, that.name) &&
Objects.equals(birthdate, that.birthdate);
51     }
52
53     @Override
54     public int hashCode() {
55         return Objects.hash(id, name, birthdate);
56     }
57
58     @Override
59     public String toString() {
60         return id + ", " + name + ", " + birthdate;
61     }
62 }
```

Листинг 1.26: OwnersCats.java

```
1 package dao.entities;
2
3 import javax.persistence.*;
4 import java.util.Objects;
5
6 @Entity
7 @Table(name = "ownerscats", schema = "public", catalog = "postgres")
8 public class OwnersCats {
9     @GeneratedValue(strategy = GenerationType.IDENTITY)
10    @Id
11    @Column(name = "id")
12    private long id;
13    @Basic
14    @Column(name = "owner_id")
15    private long ownerId;
16    @Basic
17    @Column(name = "cat_id")
18    private long catId;
19
20    public long getId() {
21        return id;
22    }
23
24    public void setId(long id) {
25        this.id = id;
26    }
27
28    public long getOwnerId() {
29        return ownerId;
30    }
31
32    public void setOwnerId(long ownerId) {
33        this.ownerId = ownerId;
34    }
35
36    public long getCatId() {
37        return catId;
38    }
39
40    public void setCatId(long catId) {
41        this.catId = catId;
42    }
43
44    @Override
45    public boolean equals(Object o) {
46        if (this == o) return true;
47        if (o == null || getClass() != o.getClass()) return false;
48        OwnersCats that = (OwnersCats) o;
49        return id == that.id && ownerId == that.ownerId && catId ==
```

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

Листинг 1.27: CatDAO.java

```
1 package dao.implementations;
2
3 import dao.interfaces.DAO;
4 import dao.entities.Cat;
5 import dao.tools.DAOException;
6 import dao.tools.HibernateSessionUtil;
7 import org.hibernate.HibernateException;
8 import org.hibernate.Session;
9
10 import java.util.List;
11
12 public class CatDAO implements DAO<Cat> {
13     @Override
14     public List<Cat> getAll() {
15         List<Cat> entities;
16
17         Session session = HibernateSessionUtil.getSessionFactory().
18         openSession();
19
20         session.getTransaction().begin();
21         entities = session.createQuery(
22             "select cats from Cat cats", Cat.class).getResultList
23         ();
24         session.getTransaction().commit();
25
26         session.close();
27
28         return entities;
29     }
30
31     @Override
32     public void add(Cat cat) throws DAOException {
33         try {
34             Session session = HibernateSessionUtil.getSessionFactory().
35             openSession();
36
37             session.getTransaction().begin();
38             session.save(cat);
39             session.getTransaction().commit();
40
41             session.close();
42
43         } catch (HibernateException e) {
44             throw new DAOException(e.getMessage(), e);
45         }
46     }
47
48     @Override
49     public Cat getByld(Long id) throws DAOException {
```

```
47     Cat cat;
48     try {
49         Session session = HibernateSessionUtil.getSessionFactory()
50         .openSession();
51
52         session.getTransaction().begin();
53         cat = session.get(Cat.class, id);
54         session.getTransaction().commit();
55
56         session.close();
57     } catch (HibernateException e) {
58         throw new DAOException(e.getMessage(), e);
59     }
60
61     return cat;
62 }
63
64 @Override
65 public void update(Cat cat) throws DAOException {
66     try {
67         Session session = HibernateSessionUtil.getSessionFactory()
68         .openSession();
69
70         session.getTransaction().begin();
71         session.update(cat);
72         session.getTransaction().commit();
73
74         session.close();
75     } catch (HibernateException e) {
76         throw new DAOException(e.getMessage(), e);
77     }
78 }
79
80 @Override
81 public void delete(Cat cat) throws DAOException {
82     try {
83         Session session = HibernateSessionUtil.getSessionFactory()
84         .openSession();
85
86         session.getTransaction().begin();
87         session.delete(cat);
88         session.getTransaction().commit();
89
90         session.close();
91     } catch (HibernateException e) {
92         throw new DAOException(e.getMessage(), e);
93     }
```

94
95

}
}

ЛИСТИНГ 1.28: CatsFriendsDAO.java

```
1 package dao.implementations;
2
3 import dao.entities.CatsFriends;
4 import dao.interfaces.DAO;
5 import dao.tools.DAOException;
6 import dao.tools.HibernateSessionUtil;
7 import org.hibernate.HibernateException;
8 import org.hibernate.Session;
9
10 import java.util.List;
11
12 public class CatsFriendsDAO implements DAO<CatsFriends> {
13     @Override
14     public List<CatsFriends> getAll() {
15         List<CatsFriends> entities;
16
17         Session session = HibernateSessionUtil.getSessionFactory().
18         openSession();
19
20         session.getTransaction().begin();
21         entities = session.createQuery(
22             "select catsfrineds from CatsFriends catsfrineds",
23             CatsFriends.class).getResultList();
24         session.getTransaction().commit();
25
26         session.close();
27
28         return entities;
29     }
30
31     @Override
32     public void add(CatsFriends catsFriends) throws DAOException {
33         try {
34             Session session = HibernateSessionUtil.getSessionFactory().
35             openSession();
36
37             session.getTransaction().begin();
38             session.save(catsFriends);
39             session.getTransaction().commit();
40
41             session.close();
42
43         } catch (HibernateException e) {
44             throw new DAOException(e.getMessage(), e);
45         }
46
47     }
48
49     @Override
50     public CatsFriends getByld(Long id) throws DAOException {
```

```
47     CatsFriends catsFriends;
48     try {
49         Session session = HibernateSessionUtil.getSessionFactory()
50         .openSession();
51
52         session.getTransaction().begin();
53         catsFriends = session.get(CatsFriends.class, id);
54         session.getTransaction().commit();
55
56         session.close();
57     } catch (HibernateException e) {
58         throw new DAOException(e.getMessage());
59     }
60
61     return catsFriends;
62 }
63
64 @Override
65 public void update(CatsFriends catsFriends) throws DAOException {
66     try {
67         Session session = HibernateSessionUtil.getSessionFactory()
68         .openSession();
69
70         session.getTransaction().begin();
71         session.update(catsFriends);
72         session.getTransaction().commit();
73
74         session.close();
75     } catch (HibernateException e) {
76         throw new DAOException(e.getMessage(), e);
77     }
78 }
79
80 @Override
81 public void delete(CatsFriends catsFriends) throws DAOException {
82     try {
83         Session session = HibernateSessionUtil.getSessionFactory()
84         .openSession();
85
86         session.getTransaction().begin();
87         session.delete(catsFriends);
88         session.getTransaction().commit();
89
90         session.close();
91     } catch (HibernateException e) {
92         throw new DAOException(e.getMessage(), e);
93     }
```

94
95

}

}

Листинг 1.29: OwnerDAO.java

```
1 package dao.implementations;
2
3 import dao.interfaces.DAO;
4 import dao.entities.Owner;
5 import dao.tools.DAOException;
6 import dao.tools.HibernateSessionUtil;
7 import org.hibernate.HibernateException;
8 import org.hibernate.Session;
9
10 import java.util.List;
11
12 public class OwnerDAO implements DAO<Owner> {
13     @Override
14     public List<Owner> getAll() {
15         List<Owner> entities;
16
17         Session session = HibernateSessionUtil.getSessionFactory().
18         openSession();
19
20         session.getTransaction().begin();
21         entities = session.createQuery(
22             "select owners from Owner owners", Owner.class).
23         getResultList();
24         session.getTransaction().commit();
25
26         session.close();
27
28         return entities;
29     }
30
31     @Override
32     public void add(Owner owner) throws DAOException {
33         try {
34             Session session = HibernateSessionUtil.getSessionFactory().
35             openSession();
36
37             session.getTransaction().begin();
38             session.save(owner);
39             session.getTransaction().commit();
40
41             session.close();
42
43         } catch (HibernateException e) {
44             throw new DAOException(e.getMessage());
45         }
46
47     }
48
49     @Override
50     public Owner getByld(Long id) throws DAOException {
```

```
47     Owner owner;
48     try {
49         Session session = HibernateSessionUtil.getSessionFactory()
50         .openSession();
51
52         session.getTransaction().begin();
53         owner = session.get(Owner.class, id);
54         session.getTransaction().commit();
55
56         session.close();
57     } catch (HibernateException e) {
58         throw new DAOException(e.getMessage());
59     }
60
61     return owner;
62 }
63
64 @Override
65 public void update(Owner owner) throws DAOException {
66     try {
67         Session session = HibernateSessionUtil.getSessionFactory()
68         .openSession();
69
70         session.getTransaction().begin();
71         session.update(owner);
72         session.getTransaction().commit();
73
74         session.close();
75     } catch (HibernateException e) {
76         throw new DAOException(e.getMessage());
77     }
78 }
79
80 @Override
81 public void delete(Owner owner) throws DAOException {
82     try {
83         Session session = HibernateSessionUtil.getSessionFactory()
84         .openSession();
85
86         session.getTransaction().begin();
87         session.delete(owner);
88         session.getTransaction().commit();
89
90         session.close();
91     } catch (HibernateException e) {
92         throw new DAOException(e.getMessage());
93     }
```

94
95

}
}

Листинг 1.30: OwnersCatsDAO.java

```
1 package dao.implementations;
2
3 import dao.entities.OwnersCats;
4 import dao.interfaces.DAO;
5 import dao.tools.DAOException;
6 import dao.tools.HibernateSessionUtil;
7 import org.hibernate.HibernateException;
8 import org.hibernate.Session;
9
10 import java.util.List;
11
12 public class OwnersCatsDAO implements DAO<OwnersCats> {
13     @Override
14     public List<OwnersCats> getAll() {
15         List<OwnersCats> entities;
16
17         Session session = HibernateSessionUtil.getSessionFactory().
18         openSession();
19
20         session.getTransaction().begin();
21         entities = session.createQuery(
22             "select ownerscats from OwnersCats ownerscats",
23             OwnersCats.class).getResultList();
24         session.getTransaction().commit();
25
26         session.close();
27
28         return entities;
29     }
30
31     @Override
32     public void add(OwnersCats ownersCats) throws DAOException {
33         try {
34             Session session = HibernateSessionUtil.getSessionFactory().
35             openSession();
36
37             session.getTransaction().begin();
38             session.save(ownersCats);
39             session.getTransaction().commit();
40
41             session.close();
42
43         } catch (HibernateException e) {
44             throw new DAOException(e.getMessage(), e);
45         }
46
47     }
48
49     @Override
50     public OwnersCats getByld(Long id) throws DAOException {
```

```
47     OwnersCats ownersCats;
48     try {
49         Session session = HibernateSessionUtil.getSessionFactory()
50         .openSession();
51
52         session.getTransaction().begin();
53         ownersCats = session.get(OwnersCats.class, id);
54         session.getTransaction().commit();
55
56         session.close();
57     } catch (HibernateException e) {
58         throw new DAOException(e.getMessage());
59     }
60
61     return ownersCats;
62 }
63
64 @Override
65 public void update(OwnersCats ownersCats) throws DAOException {
66     try {
67         Session session = HibernateSessionUtil.getSessionFactory()
68         .openSession();
69
70         session.getTransaction().begin();
71         session.update(ownersCats);
72         session.getTransaction().commit();
73
74         session.close();
75     } catch (HibernateException e) {
76         throw new DAOException(e.getMessage(), e);
77     }
78 }
79
80 @Override
81 public void delete(OwnersCats ownersCats) throws DAOException {
82     try {
83         Session session = HibernateSessionUtil.getSessionFactory()
84         .openSession();
85
86         session.getTransaction().begin();
87         session.delete(ownersCats);
88         session.getTransaction().commit();
89
90         session.close();
91     } catch (HibernateException e) {
92         throw new DAOException(e.getMessage(), e);
93     }
```


94
95

}
}

Листинг 1.31: 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> getAll();  
9     void add(T t) throws DAOException;  
10    T getByld(Long id) throws DAOException;  
11    void update(T t) throws DAOException;  
12    void delete(T t) throws DAOException;  
13 }
```

Листинг 1.32: DAOException.java

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

Листинг 1.33: HibernateSessionUtil.java

```
1 package dao.tools;
2
3 import org.hibernate.SessionFactory;
4
5 import java.io.File;
6 import org.hibernate.cfg.Configuration;
7
8 public class HibernateSessionUtil {
9     private static final SessionFactory sessionFactory =
10         initSessionFactory();
11
12     private static SessionFactory initSessionFactory() {
13         try {
14             return new Configuration().configure().buildSessionFactory
15                 ();
16         } catch (Throwable e) {
17             throw new ExceptionInInitializerError(e);
18         }
19     }
20
21     public static SessionFactory getSessionFactory() {
22         if (sessionFactory == null) {
23             initSessionFactory();
24         }
25
26         return sessionFactory;
27     }
28 }
```

Листинг 1.34: Service.java

```
1 package service;
2
3 import dao.colors.Color;
4 import dao.entities.Cat;
5 import dao.entities.CatsFriends;
6 import dao.entities.Owner;
7 import dao.entities.OwnersCats;
8 import dao.implementations.CatDAO;
9 import dao.implementations.CatsFriendsDAO;
10 import dao.implementations.OwnerDAO;
11 import dao.implementations.OwnersCatsDAO;
12 import dao.tools.DAOException;
13 import service.tools.ServiceException;
14
15 import java.sql.Timestamp;
16 import java.util.List;
17
18 public class Service {
19     private CatDAO catDAO;
20     private OwnerDAO ownerDAO;
21     private CatsFriendsDAO catsFriendsDAO;
22     private OwnersCatsDAO ownersCatsDAO;
23
24     public Service(CatDAO catDAO, OwnerDAO ownerDAO, CatsFriendsDAO
25 catsFriendsDAO, OwnersCatsDAO ownersCatsDAO) {
26         this.catDAO = catDAO;
27         this.ownerDAO = ownerDAO;
28         this.catsFriendsDAO = catsFriendsDAO;
29         this.ownersCatsDAO = ownersCatsDAO;
30     }
31
32     public Cat addCat(String name, Timestamp birthdate, String species
33 , Color color) throws ServiceException {
34         Cat cat = new Cat();
35
36         cat.setName(name);
37         cat.setBirthdate(birthdate);
38         cat.setSpecies(species);
39         cat.setColor(color);
40
41         try {
42             catDAO.add(cat);
43         } catch (DAOException e) {
44             throw new ServiceException("Problems with data access" + e
45 .getMessage());
46         }
47
48         return cat;
49     }
50 }
```

```
47
48     public void removeCat(Cat cat) throws ServiceException {
49         if (cat == null) {
50             throw new ServiceException("Cat entity can not be null!");
51         }
52
53         List<CatsFriends> catsAndFriends = catsFriendsDAO.getAll();
54
55         for (CatsFriends catAndFriend: catsAndFriends) {
56             if (catAndFriend.getFirstCatId() == cat.getId() ||
57                 catAndFriend.getSecondCatId() == cat.getId()) {
58                 try {
59                     catsFriendsDAO.delete(catAndFriend);
60                 } catch (DAOException e) {
61                     throw new ServiceException("Problem with data
62                     access" + e.getMessage());
63                 }
64             }
65
66             List<OwnersCats> ownersAndCats = ownersCatsDAO.getAll();
67
68             for (OwnersCats ownerAndCat: ownersAndCats) {
69                 if (ownerAndCat.getCatId() == cat.getId()) {
70                     try {
71                         ownersCatsDAO.delete(ownerAndCat);
72                     } catch (DAOException e) {
73                         throw new ServiceException("Problem with data
74                         access" + e.getMessage());
75                     }
76                 }
77
78                 try {
79                     catDAO.delete(cat);
80                 } catch (DAOException e) {
81                     throw new ServiceException("Problems with data access" + e
82                     .getMessage());
83                 }
84             }
85
86             public Owner addOwner(String name, Timestamp birthdate) throws
87             ServiceException {
88                 Owner owner = new Owner();
89
90                 owner.setName(name);
91                 owner.setBirthdate(birthdate);
92
93                 try {
94                     ownerDAO.add(owner);
95                 }
```

```

92         } catch (DAOException e) {
93             throw new ServiceException("Problems with data access" + e
94             .getMessage());
95         }
96         return owner;
97     }
98
99     public void removeOwner(Owner owner) throws ServiceException {
100         if (owner == null) {
101             throw new ServiceException("Owner entity can not be null!"
102             );
103         }
104         List<OwnersCats> ownersAndCats = ownersCatsDAO.getAll();
105
106         for (OwnersCats ownerAndCat: ownersAndCats) {
107             if (ownerAndCat.getOwnerId() == owner.getId()) {
108                 try {
109                     ownersCatsDAO.delete(ownerAndCat);
110                 } catch (DAOException e) {
111                     throw new ServiceException("Problem with data
112                     access" + e.getMessage());
113                 }
114             }
115
116             try {
117                 ownerDAO.delete(owner);
118             } catch (DAOException e) {
119                 throw new ServiceException("Problems with data access" + e
120                 .getMessage());
121             }
122         }
123
124         public OwnersCats addCatToOwner(Cat cat, Owner owner) throws
125         ServiceException {
126             if (cat == null) {
127                 throw new ServiceException("Cat entity can not be null!");
128             }
129
130             if (owner == null) {
131                 throw new ServiceException("Owner entity can not be null!"
132             );
133             }
134
135             List<OwnersCats> ownersAndCats = ownersCatsDAO.getAll();
136
137             for (OwnersCats ownerAndCat: ownersAndCats) {
138                 if (ownerAndCat.getOwnerId() == owner.getId() &&

```

```

136     ownerAndCat.getCatId() == cat.getId()) {
137         throw new ServiceException("Such owner-cat pair
138         already exist!");
139     }
140     OwnersCats ownersCats = new OwnersCats();
141
142     ownersCats.setOwnerId(owner.getId());
143     ownersCats.setCatId(cat.getId());
144
145     try {
146         ownersCatsDAO.add(ownersCats);
147     } catch (DAOException e) {
148         throw new ServiceException("Problems with data access" + e
149         .getMessage());
150     }
151     return ownersCats;
152 }
153
154 public void removeCatFromOwner(Cat cat, Owner owner) throws
ServiceException {
155     if (cat == null) {
156         throw new ServiceException("Cat entity can not be null!");
157     }
158
159     if (owner == null) {
160         throw new ServiceException("Owner entity can not be null!"
161         );
162     }
163
164     List<OwnersCats> ownersAndCats = ownersCatsDAO.getAll();
165
166     for (OwnersCats ownerAndCat: ownersAndCats) {
167         if (ownerAndCat.getOwnerId() == owner.getId() &&
168         ownerAndCat.getCatId() == cat.getId()) {
169             try {
170                 ownersCatsDAO.delete(ownerAndCat);
171             } catch (DAOException e) {
172                 throw new ServiceException("Problems with data
173                 access" + e.getMessage());
174             }
175         }
176     }
177
178     public CatsFriends makeFriends(Cat firstCat, Cat secondCat) throws
ServiceException {
179         if (firstCat == null) {

```



```

178         throw new ServiceException("Cat entity can not be null!");
179     }
180
181     if (secondCat == null) {
182         throw new ServiceException("Cat entity can not be null!");
183     }
184
185     CatsFriends catsFriends = new CatsFriends();
186
187     catsFriends.setFirstCatId(firstCat.getId());
188     catsFriends.setSecondCatId(secondCat.getId());
189
190     try {
191         catsFriendsDAO.add(catsFriends);
192     } catch (DAOException e) {
193         throw new ServiceException("Problems with data access" + e
194 .getMessage());
195     }
196
197     return catsFriends;
198 }
199
200 public void breakFriendship(Cat firstCat, Cat secondCat) throws
201 ServiceException {
202     if (firstCat == null) {
203         throw new ServiceException("Cat entity can not be null!");
204     }
205
206     if (secondCat == null) {
207         throw new ServiceException("Cat entity can not be null!");
208     }
209
210     List<CatsFriends> catsAndFriends = catsFriendsDAO.getAll();
211
212     for (CatsFriends catAndFriend: catsAndFriends) {
213         if (catAndFriend.getFirstCatId() == firstCat.getId() &&
214 catAndFriend.getSecondCatId() == secondCat.getId() ||
215 catAndFriend.getFirstCatId() == secondCat.getId()
216 && catAndFriend.getSecondCatId() == firstCat.getId()) {
217             try {
218                 catsFriendsDAO.delete(catAndFriend);
219             } catch (DAOException e) {
220                 throw new ServiceException("Problems with data
221 access" + e.getMessage());
222             }
223         }
224     }
225 }
226
227 public List<Cat> getAllCats() {

```

```
223         return catDAO.getAll();
224     }
225
226     public List<Owner> getAllOwners() {
227         return ownerDAO.getAll();
228     }
229
230 }
```

Листинг 1.35: ServiceException.java

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

Листинг 1.36: ServiceTest.java

```
1 package service;
2
3 import dao.colors.Color;
4 import dao.entities.Cat;
5 import dao.entities.CatsFriends;
6 import dao.entities.Owner;
7 import dao.entities.OwnersCats;
8 import dao.implementations.CatDAO;
9 import dao.implementations.CatsFriendsDAO;
10 import dao.implementations.OwnerDAO;
11 import dao.implementations.OwnersCatsDAO;
12 import org.junit.jupiter.api.Test;
13 import org.junit.jupiter.api.extension.ExtendWith;
14 import org.mockito.Mock;
15 import org.mockito.MockitoAnnotations;
16 import org.mockito.junit.jupiter.MockitoExtension;
17
18 import java.sql.Timestamp;
19 import java.util.ArrayList;
20 import java.util.List;
21
22 import static org.junit.jupiter.api.Assertions.*;
23 import static org.mockito.BDDMockito.given;
24
25 @ExtendWith(MockitoExtension.class)
26 class ServiceTest {
27
28     @Mock
29     private CatDAO catDAO;
30     @Mock
31     private OwnerDAO ownerDAO;
32     @Mock
33     private CatsFriendsDAO catsFriendsDAO;
34     @Mock
35     private OwnersCatsDAO ownersCatsDAO;
36
37     private Service service;
38
39     public ServiceTest() {
40         MockitoAnnotations.openMocks(this);
41         this.service = new Service(catDAO, ownerDAO, catsFriendsDAO,
42 ownersCatsDAO);
43     }
44
45     @Test
46     void addCat_Should_Return_True() throws Exception {
47
48         cat.setName("default");
```

```
49         cat.setBirthdate(Timestamp.valueOf("2012-12-12
00:00:00.000000000"));
50         cat.setSpecies("default");
51         cat.setColor(Color.White);
52
53         given(catDAO.getByld(1L)).willReturn(cat);
54
55         Cat returnedCat = service.addCat("default", Timestamp.valueOf(
"2012-12-12 00:00:00.000000000"),
56             "default", Color.White);
57
58         assertEquals(catDAO.getByld(1L), returnedCat);
59     }
60
61     @Test
62     void addOwner_Should_Return_True() throws Exception {
63         Owner owner = new Owner();
64
65         owner.setName("default");
66         owner.setBirthdate(Timestamp.valueOf("2012-12-12
00:00:00.000000000"));
67
68         given(ownerDAO.getByld(1L)).willReturn(owner);
69
70         Owner returnedOwner = service.addOwner("default", Timestamp.
valueOf("2012-12-12 00:00:00.000000000"));
71
72         assertEquals(ownerDAO.getByld(1L), returnedOwner);
73     }
74
75     @Test
76     void addCatToOwner_Should_Return_True() throws Exception {
77         OwnersCats ownersCats = new OwnersCats();
78
79         ownersCats.setCatId(1L);
80         ownersCats.setOwnerId(1L);
81
82         given(ownersCatsDAO.getByld(1L)).willReturn(ownersCats);
83
84         Cat cat = new Cat();
85
86         cat.setId(1L);
87         cat.setName("default");
88         cat.setBirthdate(Timestamp.valueOf("2012-12-12
00:00:00.000000000"));
89         cat.setSpecies("default");
90         cat.setColor(Color.White);
91
92         Owner owner = new Owner();
93     }
```

```
94         owner.setId(1L);
95         owner.setName("default");
96         owner.setBirthdate(Timestamp.valueOf("2012-12-12
00:00:00.000000000"));
97
98         OwnersCats returnedOwnersCats = service.addCatToOwner(cat,
owner);
99
100         assertEquals(ownersCatsDAO.getByld(1L).getCatId(),
returnedOwnersCats.getCatId());
101         assertEquals(ownersCatsDAO.getByld(1L).getOwnerId(),
returnedOwnersCats.getOwnerId());
102
103     }
104
105     @Test
106     void makeFriends_Should_Return_True() throws Exception {
107         CatsFriends catsFriends = new CatsFriends();
108
109         catsFriends.setFirstCatId(1L);
110         catsFriends.setSecondCatId(2L);
111
112         given(catsFriendsDAO.getByld(1L)).willReturn(catsFriends);
113
114         Cat cat = new Cat();
115
116         cat.setId(1L);
117         cat.setName("default");
118         cat.setBirthdate(Timestamp.valueOf("2012-12-12
00:00:00.000000000"));
119         cat.setSpecies("default");
120         cat.setColor(Color.White);
121
122         Cat cat1 = new Cat();
123
124         cat1.setId(2L);
125         cat1.setName("default");
126         cat1.setBirthdate(Timestamp.valueOf("2012-12-12
00:00:00.000000000"));
127         cat1.setSpecies("default");
128         cat1.setColor(Color.White);
129
130         CatsFriends returnedCatsFriends = service.makeFriends(cat,
cat1);
131
132         assertEquals(catsFriendsDAO.getByld(1L).getFirstCatId(),
returnedCatsFriends.getFirstCatId());
133         assertEquals(catsFriendsDAO.getByld(1L).getSecondCatId(),
returnedCatsFriends.getSecondCatId());
134     }
```

```
135
136 @Test
137 void getAllCats_Should_Return_True() {
138     given(catDAO.getAll()).willReturn(new ArrayList<>());
139
140     List<Cat> returnedList = service.getAllCats();
141
142     assertEquals(catDAO.getAll(), returnedList);
143 }
144
145 @Test
146 void getAllOwners_Should_Return_True() {
147     given(ownerDAO.getAll()).willReturn(new ArrayList<>());
148
149     List<Owner> returnedList = service.getAllOwners();
150
151     assertEquals(ownerDAO.getAll(), returnedList);
152 }
153 }
```