Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Лабороторная работа №2

Выполнил студент:

Колпикова Ксения Денисовна
группа: M32071

Проверил:

Чикишев Константин Максимович

Санкт-Петербург,
2022 г.

1.1. Текст задания

2 лабораторная
Нужно написать сервис по учету котиков и их владельцев.
Существующая информация о котиках:
- Имя
- Дата рождения
- Порода
- Цвет (один из заранее заданных вариантов)
- Хозяин
- Список котиков, с которыми дружит этот котик (из представленных в базе)

Существующая информация о хозяевах:
- Имя
- Дата рождения
- Список котиков

Сервис должен реализовывать архитектуру controller-service-dao.
Вся информация хранится в БД PostgreSQL. Для связи с БД должен использоваться Hibernate.
Проект должен собираться с помощью Maven или Gradle (на выбор студента).
Слой доступа к данным и сервисный слой должны являться двумя разными модулями Maven/Gradle. При этом проект должен полностью собираться одной командой.
При тестировании рекомендуется использовать Mockito, чтобы избежать подключения к реальным базам данных. Фреймворк для тестирования рекомендуется Junit 5.
В данной лабораторной нельзя использовать Spring или подобные ему фреймворки.

1.2. Решение

Листинг 1.1: Main.java

```java
package controller;

import DAO.enums.MyColors;
import DAO.implemetations.CatsDao;
import DAO.implemetations.FriendshipDao;
import DAO.implemetations.OwnersDao;
import DAO.implemetations.ShelterDao;
import DAO.interfaces.Dao;
import DAO.models.CatsEntity;
import DAO.models.CatsForOwnerEntity;
import DAO.models.FriendsForCatEntity;
import DAO.models.OwnersEntity;
import service.KotikiService;
import service.tools.KotikiException;

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

public class Main {
    public static void main(String args[]) throws KotikiException {
        Dao<CatsEntity> catsDao = new CatsDao();
        Dao<OwnersEntity> ownerDao = new OwnersDao();
        Dao<FriendsForCatEntity> friendshipDao = new FriendshipDao();
        Dao<CatsForOwnerEntity> shelterDao = new ShelterDao();
        KotikiService kotiki = new KotikiService(catsDao, ownerDao,
                friendshipDao, shelterDao);

        Scanner in = new Scanner(System.in);
        String str = null;

        while (!Objects.equals(str, "exit")) {
            System.out.print("Р’РІРµРґРёС‚Рµ РѕРїС†РёСЋ :" + "\n");
            System.out.print("Р”РѕР±Р°РІРёС‚СЊ РєРѕС‚Р° - 1" + "\n");
            System.out.print("Р”РѕР±Р°РІРёС‚СЊ С…РѕР·СЏРёРЅР° - 2" + "\n");
            System.out.print("РџСЂРёСЃРІРѕРёС‚СЊ РєРѕС‚Р° С…РѕР·СЏРёРЅСѓ - 3" + "\n");
            System.out.print("РџРѕРґСЂСѓР¶РёС‚СЊ РєРѕС‚РѕРІ - 4" + "\n");
            System.out.print("РЈРґР°Р»РёС‚СЊ РєРѕС‚Р° - 5" + "\n");
            System.out.print("РЈРґР°Р»РёС‚СЊ С…РѕР·СЏРёРЅР° - 6" + "\n");
            System.out.print("Р Р°Р·РѕСЂРІР°С‚СЊ РґСЂСѓР¶Р±Сѓ РєРѕС‚РѕРІ - 7" + "\n");
            System.out.print("РЎРґРµР»Р°С‚СЊ РєРѕС‚Р° Р±РµР·С…РѕР·СЏРёРЅР° - 8" + "\n");
            System.out.print("Р’С‹С…РѕРґ РІ РіР»Р°РІРЅРѕРµ РјРµРЅСЋ - exit" + "\n");
            str = in.nextLine();
            switch (str) {
                case "1":
                    CatsEntity cat = new CatsEntity();
                    System.out.print("P’PIPµPrPëC,Pµ PëPjCЏ PєPsC,P°" + "\
   n");
                    String name = in.nextLine();
                    System.out.print("P’PIPµPrPëC,Pµ PïPsCЂPsPrCѓ PєPsC,P°"
    + "\n");
                    String breed = in.nextLine();
                    System.out.print("P’PIPµPrPëC,Pµ PrP°C,Cѓ
   CЂPsP¶PrPµPSPëCЏ PєPsC,P°" + "\n");
                    System.out.print("P’ PIPëPrPµ PiPiPiPiPjPjPrPr— " + "\
   n");
                    Timestamp birthday = Timestamp.valueOf(in.nextLine
   ()+" 00:00:00");
                    System.out.print("P’PIPµPrPëC,Pµ C†PIPµC, PєPsC,P°:" +
   "\n");
                    System.out.print("Black" + "\n");
                    System.out.print("White" + "\n");
                    System.out.print("Orange" + "\n");
                    System.out.print("Brown" + "\n");
                    MyColors color = null;
                    switch (in.nextLine()) {
                        case "Black":
                            color = MyColors.Black;
                            break;
                        case "White":
                            color = MyColors.White;
                            break;
                        case "Orange":
                            color = MyColors.Orange;
                        case "Brown":
                            color = MyColors.Brown;
                            break;
                    }
                    kotiki.addCat(name, birthday, color);
                    break;
                case "2":
                    OwnersEntity owner = new OwnersEntity();
                    System.out.print("P’PIPµPrPëC,Pµ PëPjCЏ
   C...PsP·CЏPëPSP°" + "\n");
                    String nameOwner = in.nextLine();
                    System.out.print("P’PIPµPrPëC,Pµ PrP°C,Cѓ
   CЂPsP¶PrPµPSPëCЏ C...PsP·CЏPëPSP°" + "\n");
                    System.out.print("P’ PIPëPrPµ PiPiPiPiPjPjPrPr—
   C‡C‡PjPjCЃCЃ:: " + "\n");
                    Timestamp birthdayOwner = Timestamp.valueOf(in.
   nextLine());
                    kotiki.addOwner(nameOwner, birthdayOwner);
                    break;
                case "3":
                    System.out.print("P’PlPµPrPëC,Pµ id C…PsP·CЏPëPSP°" + "\n");
                    long idOwner = in.nextLong();
                    System.out.print("P’PlPµPrPëC,Pµ id PєPsC,P°" + "\n");
                    long idCat = in.nextLong();
                    kotiki.shelterCat(idOwner, idCat);
                    break;
                case "4":
                    System.out.print("P’PlPµPrPëC,Pµ id PєPsC,P°" + "\n");
                    long id1 = in.nextLong();
                    System.out.print("P’PlPµPrPëC,Pµ id PrCЂCѓPiP°" + "\n");
                    long id2 = in.nextLong();
                    kotiki.friendship(id1, id2);
                    break;
                case "5":
                    System.out.print("P’PlPµPrPëC,Pµ id PєPsC,P°" + "\n");
                    long id3 = in.nextLong();
                    kotiki.deleteCat(id3);
                    break;
                case "6":
                    System.out.print("P’PlPµPrPëC,Pµ id C…PsP·CЏPëPSP°" + "\n");
                    long id4 = in.nextLong();
                    kotiki.deleteOwner(id4);
                    break;
                case "7":
                    System.out.print("P’PlPµPrPëC,Pµ id PєPsC,P°" + "\n");
                    long id5 = in.nextLong();
                    System.out.print("P’PlPµPrPëC,Pµ id PrCЂCѓPiP°" + "\n");
                    long id6 = in.nextLong();
                    kotiki.deleteFriendship(id5, id6);
                    break;
                case "8":
                    System.out.print("P’PlPµPrPëC,Pµ id C…PsP·CЏPëPSP°" + "\n");
                    long idOwner1 = in.nextLong();
                    System.out.print("P’PlPµPrPëC,Pµ id PєPsC,P°" + "\n");
                    long idCat1 = in.nextLong();
                    kotiki.deleteShelter(idCat1, idOwner1);
                    break;
                case "exit":
                    System.out.println("finish");
                    break;
            }
        }
    }
```

```

}
```

Листинг 1.2: Bank.java

```java
package models;

import services.EventManager;
import services.IAccount;
import tools.BanksException;

import java.util.ArrayList;
import java.util.List;

public class Bank {
    public EventManager eventManager;
    private List<Client> clients;
    private List<IAccount> accountsOfClient;
    public PercentOfDeposit PercentsOfDeposit;
    public String nameOfBank;
    public double commissionOfCreditOfBank;
    public double limitOfCreditOfBank;
    public double percentOfDebitOfBank;
    public double limitationOfBank;

    public Bank(String name, double commissionOfCredit, double limitOfCredit, double percentOfDebit, double limitation,
                PercentOfDeposit percentOfDeposit) throws BanksException {
        if (percentOfDeposit == null) throw new BanksException("Invalid percents of deposit");
        PercentsOfDeposit = percentOfDeposit;
        if (name == null) throw new BanksException("Invalid name");
        nameOfBank = name;
        commissionOfCreditOfBank = commissionOfCredit;
        limitOfCreditOfBank = limitOfCredit;
        percentOfDebitOfBank = percentOfDebit;
        clients = new ArrayList<>();
        limitationOfBank = limitation;
        accountsOfClient = new ArrayList<>();
        eventManager = new EventManager("Change nameOfBank",
                "Change commissionOfCreditOfBank", "Change commissionOfCreditOfBank",
                "Change limitOfCreditOfBank", "Change percentOfDebitOfBank", "Change limitationOfBank");
    }

    public void addClientInBank(Client client) throws BanksException {
        if (client == null) throw new BanksException("Invalid client");
        if (clients.stream().anyMatch(n -> n.nameOfClient == client.nameOfClient) &&
                clients.stream().anyMatch(n -> n.surnameOfBank == client.surnameOfBank))
                throw new BanksException("Client already in use");
            clients.add(client);
        }

        public void setNameOfBank(String name) {
             this.nameOfBank = name;
             eventManager.notify("Change nameOfBank");
        }
        public String getNameOfBank() {
            return this.nameOfBank;
        }

        public void setCommissionOfCredit(double commissionOfCredit)
        {
            this.commissionOfCreditOfBank = commissionOfCredit;
            eventManager.notify("Change commissionOfCreditOfBank");
        }
        public double getCommissionOfCredit()
        {
            return this.commissionOfCreditOfBank;
        }

        public void setLimitOfCredit(double limitOfCredit)
        {
            this.limitOfCreditOfBank = limitOfCredit;
            eventManager.notify("Change limitOfCreditOfBank");
        }
        public double getLimitOfCredit()
        {
            return this.limitOfCreditOfBank;
        }

        public void setPercentOfDebit(double percentOfDebit)
        {
            this.percentOfDebitOfBank = percentOfDebit;
            eventManager.notify("Change percentOfDebitOfBank");
        }
        public double getPercentOfDebit()
        {
            return this.percentOfDebitOfBank;
        }

        public void setLimitation(double limitation)
        {
            this.limitationOfBank = limitation;
            eventManager.notify("Change limitationOfBank");
        }
        public double getLimitation()
        {
            return this.limitationOfBank;
```

```
    }

    public void addDepositInIAccount(Deposit deposit) throws BanksException {
        if (deposit == null) throw new BanksException("Invalid deposit");
        accountsOfClient.add(deposit);
    }

    public void addDebitInIAccount(Debit debit) throws BanksException {
        if (debit == null) throw new BanksException("Invalid debit");
        accountsOfClient.add(debit);
    }

    public void addCreditInIAccount(Credit credit) throws BanksException {
        if (credit == null) throw new BanksException("Invalid credit");
        accountsOfClient.add(credit);
    }

    public boolean isOperationInAccount(int sum) {
        return accountsOfClient.stream().allMatch(x -> x.withdraw(sum));
    }

    public IAccount getAccountId(int id) {
        IAccount account = null;
        for (int i = 0; i < accountsOfClient.size(); i++) {
            if (accountsOfClient.remove(i).getAccountId() == id) account = accountsOfClient.remove(i);
        }
        return account;
    }

    public int countClients() {
        return clients.size();
    }

    public int countAccounts() {
        return accountsOfClient.size();
    }
}
```

Листинг 1.3: CentralBank.java

```java
package models;

import services.IAccount;
import tools.BanksException;

import java.util.ArrayList;
import java.util.List;

public class CentralBank
{
    private static int countOfId = 0;
    private List<Bank> banks;

    public CentralBank() {

        banks = new ArrayList<>();
    }

    public void registerBank(Bank bank) throws BanksException {
        if (bank == null) throw new BanksException("Invalid Bank");
        banks.add(bank);
    }

    public IAccount creationAccount(Bank bank, int balance, Client
  client, int time, String option) throws BanksException {
        if (option == null) throw new BanksException("Invalid option")
  ;
        if (client == null) throw new BanksException("Invalid client")
  ;
        if (bank == null) throw new BanksException("Invalid Bank");
        if (banks.stream().anyMatch(b -> b == bank)) return
  creationAccountForClient(bank, client, balance, time, option);
        else throw new BanksException("Invalid bank");
    }

    public void checkTime(int time, IAccount account) throws
  BanksException {
        if (account == null) throw new BanksException("Invalid account
  ");
        account.benefitPay(time);
    }

    public IAccount creationAccountForClient(Bank bank, Client client,
    int balance, int time, String option) throws BanksException {
        switch (option)
        {
            case "Credit":

                Credit credit = new Credit(countOfId++, bank.
commissionOfCreditOfBank, bank.limitOfCreditOfBank, balance);
                bank.addCreditInIAccount(credit);
                return credit;

            case "Debit":

                Debit debit = new Debit(balance, bank.
percentOfDebitOfBank, countOfId++);
                bank.addDebitInIAccount(debit);
                return debit;

            case "Deposit":

                double depositPercent = 0;
                int f = 0;
                for ( Integer key : bank.PercentsOfDeposit.percents.
keySet() ) {
                f++;
                if (balance < key) {
                    depositPercent = bank.PercentsOfDeposit.percents.
get(key);
                }
                if( bank.PercentsOfDeposit.percents.keySet().size() -
f == 1 && depositPercent == 0)
                    depositPercent = bank.PercentsOfDeposit.percents.
get(key);
            }
                Deposit deposit = new Deposit(countOfId++, balance,
time, depositPercent);
            bank.addDepositInIAccount(deposit);
            return deposit;
            default:
                throw new BanksException("Option not found");
        }
    }

    public void refillMoneyOn(Bank bank, int balance, int id) throws
BanksException {
        if (bank == null) throw new BanksException("Invalid Bank");
        bank.getAccountId(id).refillOperation(balance);
    }

    public void withdrawalMoney(Bank bank, int balance, int id, Client
    client) throws BanksException {
        if (!checkClientPassportAndAddress(client) && balance > bank.
limitationOfBank && bank.isOperationInAccount(balance)) {
            throw new BanksException("Invalid Withdrawal");
        }
        bank.getAccountId(id).withdrawalOperation(balance);
    }

    public void transferMoneyOnBalance(Bank bank1, int balance, int id1, Bank bank2, int id2, Client client) throws BanksException {
        if (!checkClientPassportAndAddress(client) && balance > bank1.limitationOfBank && bank1.isOperationInAccount(balance)) {
            throw new BanksException("Invalid Transfer");
        }
        bank1.getAccountId(id1).transferOperation(bank2.getAccountId(id2), balance);
    }

    public void cancelRefill(Bank bank, int balance, int id) throws BanksException {
        IAccount operation = bank.getAccountId(id);
        if (operation == null) throw new BanksException("Refill don't exists");
        operation.cancelRefillOperation(balance);
    }

    public void cancelTransfer(Bank bank1, int balance, int id1, Bank bank2, int id2) throws BanksException {
        IAccount operation = bank1.getAccountId(id1);
        if (operation == null) throw new BanksException("Transfer don't exists");
        operation.cancelTransferOperation(bank2.getAccountId(id2), balance);
    }

    public void cancelWithdrawal(Bank bank, int balance, int id) throws BanksException {
        IAccount operation = bank.getAccountId(id);
        if (operation == null) throw new BanksException("Withdrawal don't exists");
        operation.cancelWithdrawalOperation(balance);
    }

    private boolean checkClientPassportAndAddress(Client client) {
        return !client.passportOfBank.isBlank() || !client.passportOfBank.isBlank();
    }
    public int countBanks(){
        return banks.size();
    }
}
```

Листинг 1.4: Client.java

```java
package models;

import tools.BanksException;

public class Client {
    private static int _id = 0;
    public int Id = 0;
    public String nameOfClient;
    public String surnameOfBank;
    public String addressOfBank;
    public String passportOfBank;

    public Client(String name, String surname, String address, String
  passport) throws BanksException {
        Id = _id++;
        if(name == null) {
            throw new BanksException("Invalid name");
        }
        nameOfClient = name;
        if(surname == null) {
            throw new BanksException("Invalid surname");
        }
        surnameOfBank = surname;
        addressOfBank = address;
        passportOfBank = passport;
    }
    public static void update(String event){
        System.out.print(event);
    }
}
```

Листинг 1.5: Credit.java

```java
package models;

import services.IAccount;

public class Credit implements IAccount {
    private double commissionTmp = 0;
    public double commissionOfCredit;
    public double limitOfBank;
    public int idOfBank;
    public int balanceOfBank;

    public Credit(int id, double commission, double limit, int balance
  )
    {
        commissionOfCredit = commission;
        limitOfBank = limit;
        idOfBank = id;
        balanceOfBank = balance;
    }

    public void withdrawalOperation(int sum) {

        balanceOfBank -=sum;
    }

    public void cancelWithdrawalOperation(int sum) {

        balanceOfBank +=sum;
    }

    public void refillOperation(int sum) {

        balanceOfBank +=sum;
    }

    public void cancelRefillOperation(int sum) {

        balanceOfBank -=sum;
    }

    public void transferOperation(IAccount other, int sum) {
        balanceOfBank -=sum;
        other.refillOperation(sum);
    }

    public void cancelTransferOperation(IAccount other, int sum) {
        balanceOfBank +=sum;
        other.withdrawalOperation(sum);
    }

    public void benefitPay(int time) {
        commissionTmp += balanceOfBank * commissionOfCredit /300;
        balanceOfBank -=(int) commissionTmp *time;
        commissionTmp =0;
    }

    public boolean withdraw(int sum) {
        return Math.abs(balanceOfBank -sum)< limitOfBank;
    }

    public int getAccountId(){

        return idOfBank;
    }

    public int checkBalance() {

        return balanceOfBank;
    }
}
```

Листинг 1.6: Debit.java

```java
package models;

import services.IAccount;

public class Debit implements IAccount {
    private double benefit = 0;
    public double Percent;
    public int Balance;
    public int Id;

    public Debit(int balance, double percent, int id) {
        Percent = percent;
        Balance = balance;
        Id = id;
    }


    public void withdrawalOperation(int sum) {

        Balance -= sum;
    }

    public void cancelWithdrawalOperation(int sum) {

        Balance += sum;
    }

    public void refillOperation(int sum) {

        Balance += sum;
    }

    public void cancelRefillOperation(int sum) {

        Balance -= sum;
    }

    public void transferOperation(IAccount account2, int sum) {
        Balance -= sum;
        account2.refillOperation(sum);
    }

    public void cancelTransferOperation(IAccount other, int sum) {
        Balance += sum;
        other.withdrawalOperation(sum);
    }

    public void benefitPay(int time) {
        benefit = Balance * Percent / 300;
```

```
            Balance += (int) benefit * time;
            benefit = 0;
        }

        public boolean withdraw(int sum) {

            return Balance >= sum;
        }

        public int getAccountId() {

            return Id;
        }

        public int checkBalance() {

            return Balance;
        }
}
```

Листинг 1.7: Deposit.java

```java
package models;


import services.IAccount;

public class Deposit implements IAccount {
    private double benefit = 0;
    public int idOfDeposit;
    public int timeOfDeposit;
    public double percentOfDeposit;
    public int balanceOfDeposit;

    public Deposit(int id, int balance, int time, double percent) {
        idOfDeposit = id;
        timeOfDeposit = time;
        percentOfDeposit = percent;
        balanceOfDeposit = balance;
    }


    public void withdrawalOperation(int sum) {

        balanceOfDeposit -= sum;
    }

    public void cancelWithdrawalOperation(int sum) {

        balanceOfDeposit += sum;
    }

    public void refillOperation(int sum) {

        balanceOfDeposit += sum;
    }

    public void cancelRefillOperation(int sum) {

        balanceOfDeposit -= sum;
    }

    public void transferOperation(IAccount other, int sum) {
        balanceOfDeposit -= sum;
        other.refillOperation(sum);
    }

    public void cancelTransferOperation(IAccount other, int sum) {
        balanceOfDeposit += sum;
        other.withdrawalOperation(sum);
    }

    public void benefitPay(int time) {
        benefit = balanceOfDeposit * percentOfDeposit / 300;
        balanceOfDeposit += (int) benefit * time;
        benefit = 0;
    }

    public boolean withdraw(int sum) {

        return balanceOfDeposit >= sum && timeOfDeposit == 0;
    }

    public int getAccountId() {

        return idOfDeposit;
    }

    public int checkBalance() {

        return balanceOfDeposit;
    }
}
```

Листинг 1.8: PercentOfDeposit.java

```
package models;

import java.util.*;

public class PercentOfDeposit
{
    public Map<Integer, Double> percents;
    public PercentOfDeposit() {

        percents = new HashMap<>();
    }

    public void addPercentAndSum(int sum, double percent)
    {
        percents.put(sum,percent);
    }
}
```

Листинг 1.9: ClientBuilder.java

```java
package services;

import models.Client;
import tools.BanksException;

public class ClientBuilder
{
    private String name;
    private String passport;
    private String surname;
    private String address;

    public ClientBuilder changeName(String name) throws BanksException {
        if(name == null) throw new BanksException("Invalid name");
        this.name = name;
        return this;
    }

    public ClientBuilder changeSurname(String surname) throws BanksException {
        if(surname == null) throw new BanksException("Invalid surname");
        this.surname = surname;
        return this;
    }

    public ClientBuilder changePassport(String passport) throws BanksException {
        if(passport == null) throw new BanksException("Invalid passport");
        this.passport = passport;
        return this;
    }

    public ClientBuilder changeAddress(String address) throws BanksException {
        if(address == null) throw new BanksException("Invalid address");
        this.address = address;
        return this;
    }

    public Client create() throws BanksException {
        return new Client(name, surname, address, passport);
    }
}
```

Листинг 1.10: EventManager.java

```java
package services;


import models.Client;

import java.util.*;

public class EventManager {
    Map<String, List<Client>> listeners = new HashMap<>();

    public EventManager(String... operations) {
        for (String operation : operations) {
            this.listeners.put(operation, new ArrayList<>());
        }
    }

    public void subscribe(String eventType, Client listener) {
        List<Client> users = listeners.get(eventType);
        users.add(listener);
    }

    public void unsubscribe(String eventType, Client listener) {
        List<Client> users = listeners.get(eventType);
        users.remove(listener);
    }

    public void notify(String event) {
        List<Client> users = listeners.get(event);
        for (Client listener : users) {
            listener.update(event);
        }
    }
}
```

Листинг 1.11: IAccount.java

```java
package services;

public interface IAccount
{
    public void withdrawalOperation(int sum);
    public void cancelWithdrawalOperation(int sum);
    public void refillOperation(int sum);
    public void cancelRefillOperation(int sum);
    public void transferOperation(IAccount other, int sum);
    public void cancelTransferOperation(IAccount other, int sum);
    public void benefitPay(int time);
    public boolean withdraw(int sum);
    public int getAccountId();
    public int checkBalance();
}
```

Листинг 1.12: BanksException.java

```java
package tools;

public class BanksException extends Exception {
    public BanksException(String message) {
        super(message);
    }
}
```

Листинг 1.13: MyColors.java

```
package DAO.enums;
public enum MyColors {
    White,
    Brown,
    Black,
    Orange
}
```

Листинг 1.14: CatsDao.java

```java
package DAO.implemetations;

import DAO.interfaces.Dao;
import DAO.models.CatsEntity;
import DAO.tools.DaoException;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import DAO.tools.HibernateUtil;

import java.util.List;

public class CatsDao implements Dao<CatsEntity> {
    @Override
    public List<CatsEntity> findAllEntity() throws DaoException {
        try {
            List<CatsEntity> entity;
            Session session = HibernateUtil.getSessionFactory().
   openSession();
            session.getTransaction().begin();
            entity = session.createQuery("select e from CatsEntity e",
    CatsEntity.class)
                    .getResultList();
            session.getTransaction().commit();
            session.close();
            return entity;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }

    @Override
    public boolean changeEntity(CatsEntity entity) throws DaoException
    {
        try {
            Session session = HibernateUtil.getSessionFactory().
   openSession();
            session.getTransaction().begin();
            session.update(entity);
            session.getTransaction().commit();
            session.close();
            return true;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }

    @Override
    public boolean addEntity(CatsEntity entity) throws DaoException {
        try {
            Session session = HibernateUtil.getSessionFactory().openSession();
            session.getTransaction().begin();
            session.save(entity);
            session.getTransaction().commit();
            session.close();
            return true;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }

    @Override
    public boolean deleteEntity(CatsEntity entity) throws DaoException {
        try {
            Session session = HibernateUtil.getSessionFactory().openSession();
            session.getTransaction().begin();
            session.delete(entity);
            session.getTransaction().commit();
            session.close();
            return true;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }

    @Override
    public CatsEntity getById(long id) throws DaoException {
        try {
            Session session = HibernateUtil.getSessionFactory().openSession();
            session.getTransaction().begin();
            CatsEntity entity = session.byId(CatsEntity.class).load(id);
            session.getTransaction().commit();
            session.close();
            return entity;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }
}
```

Листинг 1.15: FriendshipDao.java

```
package DAO.implemetations;

import DAO.interfaces.Dao;
import DAO.models.FriendsForCatEntity;
import DAO.tools.DaoException;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import DAO.tools.HibernateUtil;

import java.util.List;

public class FriendshipDao implements Dao<FriendsForCatEntity> {
    @Override
    public List<FriendsForCatEntity> findAllEntity() throws
    DaoException {
        try {
            List<FriendsForCatEntity> entity;
            Session session = HibernateUtil.getSessionFactory().
    openSession();
            session.getTransaction().begin();
            entity = session.createQuery("select e from
    FriendsForCatEntity e",
                                    FriendsForCatEntity.class)
                        .getResultList();
            session.getTransaction().commit();
            session.close();
            return entity;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }

    @Override
    public boolean changeEntity(FriendsForCatEntity entity) throws
    DaoException {
        try {
            Session session = HibernateUtil.getSessionFactory().
    openSession();
            session.getTransaction().begin();
            session.update(entity);
            session.getTransaction().commit();
            session.close();
            return true;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }

    @Override
    public boolean addEntity(FriendsForCatEntity entity) throws DaoException {
        try {
            Session session = HibernateUtil.getSessionFactory().openSession();
            session.getTransaction().begin();
            session.save(entity);
            session.getTransaction().commit();
            session.close();
            return true;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }

    @Override
    public boolean deleteEntity(FriendsForCatEntity entity) throws DaoException {
        try {
            Session session = HibernateUtil.getSessionFactory().openSession();
            session.getTransaction().begin();
            session.delete(entity);
            session.getTransaction().commit();
            session.close();
            return true;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }

    @Override
    public FriendsForCatEntity getById(long id) throws DaoException {
        try {
            Session session = HibernateUtil.getSessionFactory().openSession();
            session.getTransaction().begin();
            FriendsForCatEntity entity = session.byId(FriendsForCatEntity.class).load(id);
            session.getTransaction().commit();
            session.close();
            return entity;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }
}
```

Листинг 1.16: OwnersDao.java

```
package DAO.implemetations;

import DAO.models.OwnersEntity;
import DAO.tools.DaoException;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import DAO.interfaces.Dao;
import DAO.tools.HibernateUtil;

import java.util.List;

public class OwnersDao implements Dao<OwnersEntity> {
    @Override
    public List<OwnersEntity> findAllEntity() throws DaoException {
        try {
            List<OwnersEntity> entity;
            Session session = HibernateUtil.getSessionFactory().
    openSession();
            session.getTransaction().begin();
            entity = session.createQuery("select e from OwnersEntity e
    ", OwnersEntity.class)
                    .getResultList();
            session.getTransaction().commit();
            session.close();
            return entity;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }

    @Override
    public boolean changeEntity(OwnersEntity entity) throws
    DaoException {
        try {
            Session session = HibernateUtil.getSessionFactory().
    openSession();
            session.getTransaction().begin();
            session.update(entity);
            session.getTransaction().commit();
            session.close();
            return true;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }

    @Override
    public boolean addEntity(OwnersEntity entity) throws DaoException
    {
        try {
            Session session = HibernateUtil.getSessionFactory().
    openSession();
            session.getTransaction().begin();
            session.save(entity);
            session.getTransaction().commit();
            session.close();
            return true;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }

    @Override
    public boolean deleteEntity(OwnersEntity entity) throws
    DaoException {
        try {
            Session session = HibernateUtil.getSessionFactory().
    openSession();
            session.getTransaction().begin();
            session.delete(entity);
            session.getTransaction().commit();
            session.close();
            return true;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }

    @Override
    public OwnersEntity getById(long id) throws DaoException {
        try {
            Session session = HibernateUtil.getSessionFactory().
    openSession();
            session.getTransaction().begin();
            OwnersEntity entity = session.byId(OwnersEntity.class).
    load(id);
            session.getTransaction().commit();
            session.close();
            return entity;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }
}
```

Листинг 1.17: ShelterDao.java

```java
package DAO.implemetations;

import DAO.interfaces.Dao;
import DAO.models.CatsForOwnerEntity;
import DAO.tools.DaoException;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import DAO.tools.HibernateUtil;

import java.util.List;

public class ShelterDao implements Dao<CatsForOwnerEntity> {
    @Override
    public List<CatsForOwnerEntity> findAllEntity() throws
  DaoException {
        try {
            List<CatsForOwnerEntity> entity;
            Session session = HibernateUtil.getSessionFactory().
  openSession();
            session.getTransaction().begin();
            entity = session.createQuery("select e from
  CatsForOwnerEntity e",
                                CatsForOwnerEntity.class)
                    .getResultList();
            session.getTransaction().commit();
            session.close();
            return entity;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }

    @Override
    public boolean changeEntity(CatsForOwnerEntity entity) throws
  DaoException {
        try {
            Session session = HibernateUtil.getSessionFactory().
  openSession();
            session.getTransaction().begin();
            session.update(entity);
            session.getTransaction().commit();
            session.close();
            return true;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }

    @Override
    public boolean addEntity(CatsForOwnerEntity entity) throws DaoException {
        try {
            Session session = HibernateUtil.getSessionFactory().openSession();
            session.getTransaction().begin();
            session.save(entity);
            session.getTransaction().commit();
            session.close();
            return true;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }

    @Override
    public boolean deleteEntity(CatsForOwnerEntity pets) throws DaoException {
        try {
            Session session = HibernateUtil.getSessionFactory().openSession();
            session.getTransaction().begin();
            session.delete(pets);
            session.getTransaction().commit();
            session.close();
            return true;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }

    @Override
    public CatsForOwnerEntity getById(long id) throws DaoException {
        try {
            Session session = HibernateUtil.getSessionFactory().openSession();
            session.getTransaction().begin();
            CatsForOwnerEntity entity = session.byId(CatsForOwnerEntity.class).load(id);
            session.getTransaction().commit();
            session.close();
            return entity;
        } catch (HibernateException e) {
            throw new DaoException(e.getMessage(), e);
        }
    }
}
```

Листинг 1.18: Dao.java

```java
package DAO.interfaces;

import DAO.tools.DaoException;

import java.util.List;

public interface Dao<T> {
    List<T> findAllEntity() throws DaoException;

    boolean changeEntity(T entity) throws DaoException;

    boolean addEntity(T entity) throws DaoException;

    boolean deleteEntity(T entity) throws DaoException;

    T getById(long entity) throws DaoException;
}
```

Листинг 1.19: CatsEntity.java

```java
package DAO.models;

import DAO.enums.MyColors;

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

@Entity
@Table(name = "cats", schema = "public", catalog = "postgres")
public class CatsEntity {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id", nullable = false)
    private long id;
    @Basic
    @Column(name = "name", nullable = true, length = -1)
    private String name;
    @Basic
    @Column(name = "birth", nullable = true)
    private Timestamp birth;
    @Basic
    @Column(name = "breed", nullable = true, length = -1)
    private String breed;
    @Basic
    @Column(name = "color", nullable = true, length = -1)
    @Enumerated(EnumType.STRING)
    private MyColors color;

    public long getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

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

    public Timestamp getBirth() {
        return birth;
    }

    public void setBirth(Timestamp birth) {
        this.birth = birth;
    }

    public String getBreed() {
        return breed;
    }

    public void setBreed(String breed) {
        this.breed = breed;
    }

    public MyColors getColor() {
        return color;
    }

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

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        CatsEntity that = (CatsEntity) o;
        return id == that.id && Objects.equals(name, that.name) &&
   Objects.equals(birth, that.birth) && Objects.equals(breed, that.
   breed) && Objects.equals(color, that.color);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, name, birth, breed, color);
    }
}
```

Листинг 1.20: CatsForOwnerEntity.java

```java
package DAO.models;

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

@Entity
@Table(name = "catsForOwner", schema = "public", catalog = "postgres")
public class CatsForOwnerEntity {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id", nullable = false)
    private long id;
    @Basic
    @Column(name = "idCat", nullable = false)
    private long idCat;
    @Basic
    @Column(name = "idOwner", nullable = false)
    private long idOwner;

    public long getId() {
        return id;
    }

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

    public long getIdCat() {
        return idCat;
    }

    public void setIdCat(long idCat) {
        this.idCat = idCat;
    }

    public long getIdOwner() {
        return idOwner;
    }

    public void setIdOwner(long idOwner) {
        this.idOwner = idOwner;
    }


    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        CatsForOwnerEntity that = (CatsForOwnerEntity) o;
        return id == that.id && idCat == that.idCat && idOwner == that
    .idOwner;
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, idCat, idOwner);
    }
}
```

Листинг 1.21: FriendsForCatEntity.java

```java
package DAO.models;

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

@Entity
@Table(name = "friendsForCat", schema = "public", catalog = "postgres"
    )
public class FriendsForCatEntity {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id", nullable = false)
    private long id;
    @Basic
    @Column(name = "idFriend", nullable = false)
    private long idFriend;
    @Basic
    @Column(name = "idCat", nullable = false)
    private long idCat;

    public long getId() {
        return id;
    }

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

    public long getIdFriend() {
        return idFriend;
    }

    public void setIdFriend(long idFriend) {
        this.idFriend = idFriend;
    }

    public long getIdCat() {
        return idCat;
    }

    public void setIdCat(long idCat) {
        this.idCat = idCat;
    }



    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        FriendsForCatEntity that = (FriendsForCatEntity) o;
        return id == that.id && idFriend == that.idFriend && idCat ==
    that.idCat;
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, idFriend, idCat);
    }
}
```

Листинг 1.22: OwnersEntity.java

```java
package DAO.models;

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

@Entity
@Table(name = "owners", schema = "public", catalog = "postgres")
public class OwnersEntity {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id", nullable = false)
    private long id;
    @Basic
    @Column(name = "name", nullable = true, length = -1)
    private String name;
    @Basic
    @Column(name = "date", nullable = true)
    private Timestamp date;

    public long getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

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

    public Timestamp getDate() {
        return date;
    }

    public void setDate(Timestamp date) {
        this.date = date;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        OwnersEntity that = (OwnersEntity) o;
        return id == that.id && Objects.equals(name, that.name) &&
    Objects.equals(date, that.date);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, name, date);
    }
}
```

Листинг 1.23: DaoException.java

```java
package DAO.tools;

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

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

    public DaoException(String message, Throwable cause) {
        super(message, cause);

    }
}
```

Листинг 1.24: HibernateUtil.java

```java
package DAO.tools;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

import java.io.File;

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

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

    public static SessionFactory getSessionFactory() {

        if (sessionFactory == null){
            initSessionFactory();
        }

        return sessionFactory;
    }

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

}
```

Листинг 1.25: KotikiService.java

```java
package service;

import DAO.enums.MyColors;
import DAO.interfaces.Dao;
import DAO.models.CatsEntity;
import DAO.models.CatsForOwnerEntity;
import DAO.models.FriendsForCatEntity;
import DAO.models.OwnersEntity;
import DAO.tools.DaoException;
import service.tools.KotikiException;

import java.sql.Timestamp;

public class KotikiService {

    private final Dao<CatsEntity> catsDao;
    private final Dao<OwnersEntity> ownerDao;
    private final Dao<FriendsForCatEntity> friendshipDao;
    private final Dao<CatsForOwnerEntity> shelterDao;


    public KotikiService(Dao<CatsEntity> catsDao, Dao<OwnersEntity>
    ownerDao,
                         Dao<FriendsForCatEntity> friendshipDao,
                         Dao<CatsForOwnerEntity> shelterDao) {
        this.catsDao = catsDao;
        this.ownerDao = ownerDao;
        this.friendshipDao = friendshipDao;
        this.shelterDao = shelterDao;
    }


    public boolean deleteCat(long idCat) throws KotikiException {
        if (idCat == 0) throw new KotikiException("Invalid cat");
        try {
            for (FriendsForCatEntity fr : friendshipDao.findAllEntity
    ()) {
                if (fr.getIdCat() == idCat ||
                        fr.getIdFriend() == idCat) friendshipDao.
    deleteEntity(fr);
            }
        } catch (DaoException e) {
         throw new KotikiException(e.getMessage());
        }

        try {
            return catsDao.deleteEntity(catsDao.getById(idCat));
        } catch (DaoException e) {
            throw new KotikiException(e.getMessage());
        }
    }

    public boolean deleteOwner(long idOwner) throws KotikiException {
        if (idOwner == 0) throw new KotikiException("Invalid owner");
        try {
            for (CatsForOwnerEntity fr : shelterDao.findAllEntity()) {
                if (fr.getIdOwner() == idOwner) shelterDao.
   deleteEntity(fr);
            }
        } catch (DAO.tools.DaoException e) {
            throw new KotikiException(e.getMessage());
        }
        try {
            return ownerDao.deleteEntity(ownerDao.getById(idOwner));
        } catch (DAO.tools.DaoException e) {
            throw new KotikiException(e.getMessage());
        }
    }

    public boolean changeCat(long idCat) throws KotikiException {
        if (idCat == 0) throw new KotikiException("Invalid cat");
        try {
            return catsDao.changeEntity(catsDao.getById(idCat));
        } catch (DAO.tools.DaoException e) {
            throw new KotikiException(e.getMessage());
        }
    }

    public boolean changeOwner(long owner) throws KotikiException {
        if (owner == 0) throw new KotikiException("Invalid owner");
        try {
            return ownerDao.changeEntity(ownerDao.getById(owner));
        } catch (DAO.tools.DaoException e) {
            throw new KotikiException(e.getMessage());
        }
    }

    public boolean addOwner(String name, Timestamp birthday) throws
   KotikiException {
        OwnersEntity owner = new OwnersEntity();
        owner.setName(name);
        owner.setDate(birthday);
        try {
            return ownerDao.addEntity(owner);
        } catch (DAO.tools.DaoException e) {
            throw new KotikiException(e.getMessage());
        }
    }

    public boolean addCat(String name, Timestamp birthday, MyColors
   color) throws KotikiException {
        CatsEntity cat = new CatsEntity();
        cat.setColor(color);
        cat.setName(name);
        cat.setBirth(birthday);
        try {
            return catsDao.addEntity(cat);
        } catch (DAO.tools.DaoException e) {
            throw new KotikiException(e.getMessage());
        }
    }

    public boolean friendship(long cat1, long cat2) throws
   KotikiException {
        if (cat1 == 0) throw new KotikiException("Invalid cat1");
        if (cat2 == 0) throw new KotikiException("Invalid cat2");
        FriendsForCatEntity friends = new FriendsForCatEntity();
        friends.setIdCat(cat1);
        friends.setIdFriend(cat2);

        try {
            return friendshipDao.addEntity(friends);
        } catch (DAO.tools.DaoException e) {
            throw new KotikiException(e.getMessage());
        }
    }

    public boolean shelterCat(long owner, long cat) throws
   KotikiException {
        if (cat == 0) throw new KotikiException("Invalid cat");
        if (owner == 0) throw new KotikiException("Invalid owner");
        try {
            for (CatsForOwnerEntity pt : shelterDao.findAllEntity()) {
                if (pt.getIdCat() == cat) return false;
            }
        } catch (DAO.tools.DaoException e) {
            throw new KotikiException(e.getMessage());
        }
        CatsForOwnerEntity pets = new CatsForOwnerEntity();
        pets.setIdCat(cat);
        pets.setIdOwner(owner);

        try {
            return shelterDao.addEntity(pets);
        } catch (DAO.tools.DaoException e) {
            throw new KotikiException(e.getMessage());
        }
    }

```

```java
    public boolean deleteFriendship(long id1, long id2) throws KotikiException {
        if (id1 == id2) throw new KotikiException("Invalid friends");
        try {
            for (FriendsForCatEntity fr : friendshipDao.findAllEntity()) {
                if (fr.getIdCat() == id1 && fr.getIdFriend() == id2 ||
                        fr.getIdFriend() == id1 && fr.getIdCat() == id2) return friendshipDao.deleteEntity(fr);
            }
        } catch (DAO.tools.DaoException e) {
            throw new KotikiException(e.getMessage());
        }
        return false;
    }

    public boolean deleteShelter(long idCat, long idOwner) throws KotikiException {
        try {
            for (CatsForOwnerEntity fr : shelterDao.findAllEntity()) {
                if (fr.getIdCat() == idCat && fr.getIdOwner() == idOwner) shelterDao.deleteEntity(fr);

            }
        } catch (DAO.tools.DaoException e) {
            throw new KotikiException(e.getMessage());
        }
        return true;
    }
}
```

Листинг 1.26: KotikiException.java

```java
package service.tools;

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

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

    public KotikiException(String message, Throwable cause) {
        super(message, cause);

    }
}
```

Листинг 1.27: KotikiServiceTest.java

```
package service;

import DAO.enums.MyColors;
import DAO.interfaces.Dao;
import DAO.models.CatsEntity;
import DAO.models.CatsForOwnerEntity;
import DAO.models.FriendsForCatEntity;
import DAO.models.OwnersEntity;
import org.junit.jupiter.api.Test;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import service.tools.KotikiException;

import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.BDDMockito.*;

class KotikiServiceTest {
    @Mock
    private  Dao<CatsEntity> catsDao;
    @Mock
    private  Dao<OwnersEntity> ownerDao;
    @Mock
    private  Dao<FriendsForCatEntity> friendshipDao;
    @Mock
    private  Dao<CatsForOwnerEntity> shelterDao;
    private KotikiService kotikiService;

    public KotikiServiceTest() {
        MockitoAnnotations.initMocks(this);
        this.kotikiService = new KotikiService(catsDao, ownerDao,
   friendshipDao, shelterDao);
    }


    @Test
    void deleteCat() throws KotikiException {
        CatsEntity cat = new CatsEntity();
        FriendsForCatEntity friends = new FriendsForCatEntity();
        List<FriendsForCatEntity> listFriends = new ArrayList<>();
        try {
            given(friendshipDao.findAllEntity()).willReturn(
   listFriends);
        } catch (DAO.tools.DaoException e) {
            e.printStackTrace();
        }
        try {
            given(friendshipDao.deleteEntity(friends)).willReturn(true);
        } catch (DAO.tools.DaoException e) {
            e.printStackTrace();
        }
        try {
            given(catsDao.deleteEntity(cat)).willReturn(true);
        } catch (DAO.tools.DaoException e) {
            e.printStackTrace();
        }
        try {
            given(catsDao.getById(1)).willReturn(cat);
        } catch (DAO.tools.DaoException e) {
            e.printStackTrace();
        }
        boolean exist = kotikiService.deleteCat(1);
        assertTrue(exist);
    }

    @Test
    void addCat() throws KotikiException {
        CatsEntity cat = new CatsEntity();
        cat.setBirth(Timestamp.valueOf("2002-07-13 00:00:00"));
        cat.setName("nice");
        cat.setColor(MyColors.Black);
        try {
            given(catsDao.addEntity(cat)).willReturn(true);
        } catch (DAO.tools.DaoException e) {
            e.printStackTrace();
        }
        boolean exist = kotikiService.addCat("nice", Timestamp.valueOf("2002-07-13 00:00:00"), MyColors.Black);
        assertTrue(exist);
    }

    @Test
    void friendship() throws KotikiException {
        CatsEntity cat = new CatsEntity();
        CatsEntity cat1 = new CatsEntity();
        cat.setName("hfhf");
        cat1.setName("hkh");
        FriendsForCatEntity friends = new FriendsForCatEntity();
        friends.setIdCat(1);
        friends.setIdFriend(2);
        try {
            given(friendshipDao.addEntity(friends)).willReturn(true);
        } catch (DAO.tools.DaoException e) {
            e.printStackTrace();
        }
        boolean exist = kotikiService.friendship(1,2);
        assertTrue(exist);
    }

    @Test
    void shelterCat() throws KotikiException {
            CatsEntity cat = new CatsEntity();
            OwnersEntity own = new OwnersEntity();
            CatsForOwnerEntity pets = new CatsForOwnerEntity();
            pets.setIdCat(1);
            pets.setIdOwner(2);
        try {
            given(shelterDao.addEntity(pets)).willReturn(true);
        } catch (DAO.tools.DaoException e) {
            e.printStackTrace();
        }
        boolean exist = kotikiService.shelterCat(2,1);
            assertTrue(exist);
    }

    @Test
    void deleteFriendship() throws KotikiException {

        FriendsForCatEntity friends = new FriendsForCatEntity();
        friends.setIdFriend(1);
        friends.setIdCat(2);
        List<FriendsForCatEntity> listFriends = new ArrayList<>();
        listFriends.add(friends);
        try {
            given(friendshipDao.findAllEntity()).willReturn(
    listFriends);
        } catch (DAO.tools.DaoException e) {
            e.printStackTrace();
        }
        try {
            given(friendshipDao.deleteEntity(friends)).willReturn(true
    );
        } catch (DAO.tools.DaoException e) {
            e.printStackTrace();
        }
        boolean exist = kotikiService.deleteFriendship(1,2);
        assertTrue(exist);
    }
}
```