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:

Жарков Григорий Алексеевич
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: Console.java

```java
package controller;

import controller.tools.ControllerException;
import dao.colors.Color;
import dao.entities.Cat;
import dao.entities.Owner;
import service.Service;
import service.tools.ServiceException;

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

public class Console {
    private Service service;

    public Console(Service service) {
        this.service = service;
    }

    public void work() throws ControllerException {
        while (true) {
            System.out.println("""
                    Enter what you want to do:\s
                    1) Add new cat\s
                    2) Add new owner\s
                    3) Add cat to owner\s
                    4) Remove cat from owner\s
                    5) Make friends\s
                    6) Break friendship\s
                    7) Remove cat\s
                    8) Remove owner\s
                    0) Exit""");

            Scanner choice = new Scanner(System.in);
            String answer = choice.nextLine();

            switch (answer) {
                case "1" -> {
                    System.out.println("Enter cat's name:");
                    String name = choice.nextLine();

                    System.out.println("Enter cat's birthdate (yyyy-mm-dd):");
                    String birthdate = choice.nextLine();

                    System.out.println("Enter cat's species:");
                    String species = choice.nextLine();

                System.out.println("Enter cat's color (Black, White, Gray or Orange):");
                String color = choice.nextLine();

                birthdate = birthdate.concat(" 00:00:00");

                try {
                    service.addCat(name, Timestamp.valueOf(birthdate), species, Color.valueOf(color));
                } catch (ServiceException e) {
                    throw new ControllerException("Problems with adding cat to service" + e.getMessage());
                }
            }

            case "2" -> {
                System.out.println("Enter owner's name:");
                String name = choice.nextLine();

                System.out.println("Enter owner's birthdate (yyyy-mm-dd):");
                String birthdate = choice.nextLine();

                birthdate = birthdate.concat(" 00:00:00");

                try {
                    service.addOwner(name, Timestamp.valueOf(birthdate));
                } catch (ServiceException e) {
                    throw new ControllerException("Problems with adding owner to service" + e.getMessage());
                }
            }

            case "3" -> {
                List<Cat> cats = service.getAllCats();
                List<Owner> owners = service.getAllOwners();

                System.out.println("Choose one cat to add to owner:");
                for (Cat cat: cats) {
                    System.out.println(cat);
                }
                String catNumber = choice.nextLine();

                if (Integer.parseInt(catNumber) >= cats.size()) {
                    throw new ArrayIndexOutOfBoundsException();
                }

                System.out.println("Choose one owner to add cat to:");
                for (Owner owner: owners) {
                    System.out.println(owner);
                }
                String ownerNumber = choice.nextLine();

                if (Integer.parseInt(ownerNumber) >= owners.size()) {
                    throw new ArrayIndexOutOfBoundsException();
                }

                try {
                    service.addCatToOwner(cats.get(Integer.parseInt(catNumber)), owners.get(Integer.parseInt(ownerNumber)));
                } catch (ServiceException e) {
                    throw new ControllerException("Problems with adding cat to service" + e.getMessage());
                }
            }

            case "4" -> {
                List<Cat> cats = service.getAllCats();
                List<Owner> owners = service.getAllOwners();

                System.out.println("Choose one cat to be removed from owner:");
                for (Cat cat: cats) {
                    System.out.println(cat);
                }
                String catNumber = choice.nextLine();

                if (Integer.parseInt(catNumber) >= cats.size()) {
                    throw new ArrayIndexOutOfBoundsException();
                }

                System.out.println("Choose one owner whose cat will be removed:");
                for (Owner owner: owners) {
                    System.out.println(owner);
                }
                String ownerNumber = choice.nextLine();

                if (Integer.parseInt(ownerNumber) >= owners.size()) {
                    throw new ArrayIndexOutOfBoundsException();
                }

                try {
```

```java
                        service.removeCatFromOwner(cats.get(Integer.parseInt(catNumber)), owners.get(Integer.parseInt(ownerNumber)));
                    } catch (ServiceException e) {
                        throw new ControllerException("Problems with removing cat from owner" + e.getMessage());
                    }
                }

                case "5" -> {
                    List<Cat> cats = service.getAllCats();

                    System.out.println("Choose first cat:");
                    for (Cat cat: cats) {
                        System.out.println(cat);
                    }
                    String firstCatNumber = choice.nextLine();

                    if (Integer.parseInt(firstCatNumber) >= cats.size()) {
                        throw new ArrayIndexOutOfBoundsException();
                    }

                    System.out.println("Choose second cat:");
                    for (Cat cat: cats) {
                        System.out.println(cat);
                    }
                    String secondCatNumber = choice.nextLine();

                    if (Integer.parseInt(secondCatNumber) >= cats.size()) {
                        throw new ArrayIndexOutOfBoundsException();
                    }

                    try {
                        service.makeFriends(cats.get(Integer.parseInt(firstCatNumber)), cats.get(Integer.parseInt(secondCatNumber)));
                    } catch (ServiceException e) {
                        throw new ControllerException("Problems with making friendship" + e.getMessage());
                    }
                }

                case "6" -> {
                    List<Cat> cats = service.getAllCats();

                    System.out.println("Choose first cat:");
                    for (Cat cat: cats) {
                        System.out.println(cat);
                    }
                    String firstCatNumber = choice.nextLine();

                    if (Integer.parseInt(firstCatNumber) >= cats.size()) {
                        throw new ArrayIndexOutOfBoundsException();
                    }

                    System.out.println("Choose second cat:");
                    for (Cat cat: cats) {
                        System.out.println(cat);
                    }
                    String secondCatNumber = choice.nextLine();

                    if (Integer.parseInt(secondCatNumber) >= cats.size()) {
                        throw new ArrayIndexOutOfBoundsException();
                    }

                    try {
                        service.breakFriendship(cats.get(Integer.parseInt(firstCatNumber)), cats.get(Integer.parseInt(secondCatNumber)));
                    } catch (ServiceException e) {
                        throw new ControllerException("Problems with breaking friendship" + e.getMessage());
                    }
                }

                case "7" -> {
                    List<Cat> cats = service.getAllCats();

                    System.out.println("Choose cat to be removed:");
                    for (Cat cat: cats) {
                        System.out.println(cat);
                    }
                    String catNumber = choice.nextLine();

                    if (Integer.parseInt(catNumber) >= cats.size()) {
                        throw new ArrayIndexOutOfBoundsException();
                    }

                    try {
                        service.removeCat(cats.get(Integer.parseInt(catNumber)));
                    } catch (ServiceException e) {
                        throw new ControllerException("Problems with removing cat from service" + e.getMessage());
                    }
                }

                case "8" -> {
                    List<Owner> owners = service.getAllOwners();

                    System.out.println("Choose owner to be removed:");
                    for (Owner owner: owners) {
                        System.out.println(owner);
                    }
                    String ownerNumber = choice.nextLine();

                    if (Integer.parseInt(ownerNumber) >= owners.size()) {
                        throw new ArrayIndexOutOfBoundsException();
                    }

                    try {
                        service.removeOwner(owners.get(Integer.parseInt(ownerNumber)));
                    } catch (ServiceException e) {
                        throw new ControllerException("Problems with removing owner from service" + e.getMessage());
                    }
                }

                case "0" -> System.exit(0);

                default -> {
                    var e = new IllegalArgumentException();
                    throw new ControllerException("Invalid option number!", e);
                }
            }
        }
    }
}
```

Листинг 1.2: Main.java

```java
package controller;

import controller.tools.ControllerException;
import dao.implementations.CatDAO;
import dao.implementations.CatsFriendsDAO;
import dao.implementations.OwnerDAO;
import dao.implementations.OwnersCatsDAO;
import service.Service;

public class Main {
    public static void main(String[] args) throws ControllerException
   {
        Service service = new Service(new CatDAO(), new OwnerDAO(),
   new CatsFriendsDAO(), new OwnersCatsDAO());
        Console ui = new Console(service);
        ui.work();
    }
}
```

Листинг 1.3: AccountType.java

```
package account;

public enum AccountType {
    Debit,
    Deposit,
    Credit,
}
```

Листинг 1.4: Credit.java

```java
package account;

import tools.BankException;

import java.util.UUID;

public class Credit implements IAccount {
    private double fee;
    private double limit;

    private double unverifiedLimit;
    private boolean verified;

    private double accumulatedFee;
    private double balance;

    private UUID id;

    public Credit(double fee, double limit, boolean verified, double unverifiedLimit) throws BankException {
        if (fee <= 0) throw new BankException("Fee for credit account must be positive!");
        if (limit <= 0) throw new BankException("Limit for credit account must be positive!");
        if (unverifiedLimit <= 0) throw new BankException("Limit for unverified account must be positive!");

        this.fee = fee;
        this.limit = limit;
        this.unverifiedLimit = unverifiedLimit;
        this.verified = verified;

        id = UUID.randomUUID();

        accumulatedFee = 0;
        balance = 0;
    }

    @Override
    public UUID getId() {
        return id;
    }

    @Override
    public double getBalance() {
        return balance;
    }

    public void takeMoney(double amount) throws BankException {
        if (amount <= 0) throw new BankException("Amount must be positive!");
        if (amount > balance + limit) throw new BankException("Amount is too big!");
        if (amount > unverifiedLimit && !verified)
            throw new BankException("Amount is bigger than limit for unverified account!");

        balance -= amount;
    }

    public void addMoney(double amount) throws BankException {
        if (amount <= 0) throw new BankException("Amount must be positive!");
        balance += amount;
    }

    public void calculateDailyPayment() {
        if (balance < 0) {
            accumulatedFee += fee;
        }
    }

    public void getReward() throws BankException {
        if (balance + limit < accumulatedFee) throw new BankException("Balance too low to take commission!");
        balance -= accumulatedFee;
        accumulatedFee = 0;
    }

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

        Credit other = (Credit) obj;

        return other.getId() == this.getId();
    }

    @Override
    public int hashCode() {
        return id.hashCode();
    }
}
```

Листинг 1.5: Debit.java

```java
package account;

import tools.BankException;

import java.util.UUID;

public class Debit implements IAccount {
    private double interest;

    private double unverifiedLimit;
    private boolean verified;

    private double accumulatedAmount;
    private double balance;

    private UUID id;

    public Debit(double interest, boolean verified, double unverifiedLimit) throws BankException {
        if (interest <= 0) throw new BankException("Interest for debit account must be positive!");
        if (unverifiedLimit <= 0) throw new BankException("Limit for unverified account must be positive!");

        this.interest = interest;
        this.verified = verified;
        this.unverifiedLimit = unverifiedLimit;

        id = UUID.randomUUID();

        accumulatedAmount = 0;
        balance = 0;
    }

    @Override
    public UUID getId() {
        return id;
    }

    @Override
    public double getBalance() {
        return balance;
    }

    public void takeMoney(double amount) throws BankException {
        if (amount <= 0) throw new BankException("Amount must be positive!");
        if (amount > balance) throw new BankException("Amount is too big!");
        if (amount > unverifiedLimit && !verified)
            throw new BankException("Amount is bigger than limit for unverified account!");

        balance -= amount;
    }

    public void addMoney(double amount) throws BankException {
        if (amount <= 0) throw new BankException("Amount must be positive!");
        balance += amount;
    }

    @Override
    public void calculateDailyPayment() {
        accumulatedAmount += balance * interest / 365;
    }

    @Override
    public void getReward() {
        balance += accumulatedAmount;
        accumulatedAmount = 0;
    }

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

        Debit other = (Debit) obj;

        return other.getId() == this.getId();
    }

    @Override
    public int hashCode() {
        return id.hashCode();
    }
}
```

Листинг 1.6: Deposit.java

```java
package account;

import tools.BankException;

import java.time.LocalDate;
import java.util.Map;
import java.util.UUID;

public class Deposit implements IAccount {
    private double interest;

    private double unverifiedLimit;
    private boolean verified;

    private LocalDate validUntil;

    private double accumulatedAmount;
    private double balance;

    private UUID id;

    public Deposit(double interest, Map<Double, Double>
   interestConditions,
                    DepositDTO depositData, boolean verified, double
   unverifiedLimit) throws BankException {
        if (interest <= 0) throw new BankException("Interest for debit
    account must be positive!");
        if (unverifiedLimit <= 0) throw new BankException("Limit for
   unverified account must be positive!");

        for (Double percent : interestConditions.keySet()) {
            if (percent <= 0) throw new BankException("Percent can not
    be null!");
        }

        for (Double amount : interestConditions.values()) {
            if (amount <= 0) throw new BankException("Amount can not
   be null!");
        }

        if (depositData == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Deposit data can not be null!", e
   );
        }

        balance = depositData.getBalance();
        validUntil = LocalDate.parse(depositData.getValidUntil());

        if (balance <= 0) throw new BankException("Balance must be positive!");
        if (validUntil.isBefore(LocalDate.now())) throw new BankException("Invalid date!");

        this.interest = interest;
        for (Map.Entry<Double, Double> condition : interestConditions.entrySet()) {
            if (balance <= condition.getKey()) this.interest = condition.getValue();
        }

        this.verified = verified;
        this.unverifiedLimit = unverifiedLimit;

        id = UUID.randomUUID();
        accumulatedAmount = 0;
    }

    @Override
    public UUID getId() {
        return id;
    }

    @Override
    public double getBalance() {
        return balance;
    }

    @Override
    public void takeMoney(double amount) throws BankException {
        if (amount <= 0) throw new BankException("Amount must be positive!");
        if (amount > balance) throw new BankException("Amount is too big!");
        if (amount > unverifiedLimit && !verified)
            throw new BankException("Amount is bigger than limit for unverified account!");

        if (LocalDate.now().isBefore(validUntil)) throw new BankException("It is impossible to take money now!");

        balance -= amount;
    }

    @Override
    public void addMoney(double amount) throws BankException {
        if (amount <= 0) throw new BankException("Amount must be positive!");
        balance += amount;
    }

    @Override
    public void calculateDailyPayment() {
        accumulatedAmount += balance * interest / 365;
    }

    @Override
    public void getReward() {
        balance += accumulatedAmount;
        accumulatedAmount = 0;
    }

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

        Deposit other = (Deposit) obj;

        return other.getId() == this.getId();
    }

    @Override
    public int hashCode() {
        return id.hashCode();
    }
}
```

Листинг 1.7: DepositConsole.java

```java
package account;

import java.util.Scanner;

public class DepositConsole {
    public DepositDTO collectDepositConditions() {
        Scanner in = new Scanner(System.in);

        System.out.println("Enter amount you want to deposit:");
        double balance = in.nextDouble();

        System.out.println("Enter date you want deposit will be valid
   until (dd-mm-yyyy):");
        String date = in.nextLine();

        return new DepositDTO(balance, date);
    }
}
```

Листинг 1.8: DepositDTO.java

```java
package account;

public class DepositDTO {
    private double balance;
    private String validUntil;

    public DepositDTO(double balance, String validUntil){
        this.balance = balance;
        this.validUntil = validUntil;
    }

    public double getBalance() {
        return balance;
    }

    public String getValidUntil() {
        return validUntil;
    }
}
```

Листинг 1.9: IAccount.java

```java
package account;

import tools.BankException;

import java.util.UUID;

public interface IAccount {

    double getBalance();
    UUID getId();

    void takeMoney(double amount) throws BankException;
    void addMoney(double amount) throws BankException;

    void calculateDailyPayment();
    void getReward() throws BankException;

}
```

Листинг 1.10: Bank.java

```java
package bank;

import account.*;
import client.Client;
import client.ClientDTO;
import tools.BankException;
import tools.EventManager;
import tools.IEventListener;

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

public class Bank implements IEventListener {

    private CentralBank centralBank;
    private List<Client> clients;

    private String name;
    private double debitInterest;
    private double creditFee;
    private double creditLimit;
    private double unverifiedLimit;
    private double depositDefaultInterest;
    private Map<Double, Double> depositInterestConditions;

    public EventManager events;

    public Bank(CentralBank centralBank, BankDTO bankData) throws
  BankException {
        if (centralBank == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Central bank can not be null!", e
  );
        }

        if (bankData == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Bank data can not be null!", e);
        }

        if (bankData.getName() == null) {
            var e =  new IllegalArgumentException();
            throw new BankException("Bank data can not be null!", e);
        }
        if (bankData.getCreditFee() <= 0) throw new BankException("
  Credit fee can not be negative!");
        if (bankData.getCreditLimit() <= 0) throw new BankException("
  Credit limit can not be negative!");
```

```
        if (bankData.getDebitInterest() <= 0) throw new BankException("Debit interest can not be negative!");
        if (bankData.getDepositDefaultInterest() <= 0) throw new BankException("Deposit default interest can not be negative!");
        if (bankData.getUnverifiedLimit() <= 0) throw new BankException("Unverified limit can not be negative!");

        this.centralBank = centralBank;

        name = bankData.getName();

        debitInterest = bankData.getDebitInterest();
        creditFee = bankData.getCreditFee();
        creditLimit = bankData.getCreditLimit();
        unverifiedLimit = bankData.getUnverifiedLimit();
        depositDefaultInterest = bankData.getDepositDefaultInterest();
        depositInterestConditions = bankData.getDepositInterestConditions();

        clients = new ArrayList<>();

        events = new EventManager("unverified limit", "debit interest", "credit fee", "credit limit");
    }

    public String getName() {
        return name;
    }

    public double getDebitInterest() {
        return debitInterest;
    }

    public double getCreditFee() {
        return creditFee;
    }

    public double getCreditLimit() {
        return creditLimit;
    }

    public double getUnverifiedLimit() {
        return unverifiedLimit;
    }

    public double getDepositDefaultInterest() {
        return depositDefaultInterest;
    }

    public List<Client> getClients() {
        return clients;
    }

    public Client registerClient(ClientDTO clientData) throws
   BankException {
        if (clientData == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Client data can not be null!", e)
;
        }

        for (Client client: clients) {
            if (client.getId() == clientData.getId()) {
                throw new BankException("Such client already exist!");
            }
        }

        Client client = new Client(clientData, this);
        clients.add(client);

        return client;
    }

    public Client fillMissingData(Client client, ClientDTO clientData)
    throws BankException {
        if (client == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Client can not be null!", e);
        }

        if (clientData == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Client data can not be null!", e)
;
        }

        if (!clients.contains(client)) throw new BankException("
   Unknown client!");

        client.addMissingData(clientData);

        return client;
    }

    public IAccount registerAccount(Client client, AccountType
   accountType, DepositDTO depositData) throws BankException {
        if (client == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Client can not be null!", e);
        }

        IAccount account;
        switch (accountType) {
            case Credit -> account = new Credit(creditFee, creditLimit, client.getVerified(), unverifiedLimit);
            case Debit -> account = new Debit(debitInterest, client.getVerified(), unverifiedLimit);
            case Deposit -> account = new Deposit(depositDefaultInterest, depositInterestConditions, depositData, client.getVerified(), unverifiedLimit);
            default -> {
                var e = new IllegalArgumentException();
                throw new BankException("Invalid account type!", e);
            }
        }

        client.addAccount(account);

        return account;
    }

    public void calculateDailyPayment() {
        for (Client client: clients) {
            for (IAccount account: client.getAccounts()) {
                account.calculateDailyPayment();
            }
        }
    }

    public void payReward() throws BankException {
        for (Client client: clients) {
            for (IAccount account: client.getAccounts()) {
                account.getReward();
            }
        }
    }

    public void changeUnverifiedLimit(double newLimit) throws BankException {
        if (newLimit <= 0) throw new BankException("Limit must be positive!");
        unverifiedLimit = newLimit;

        events.notify("unverified limit");
    }

    public void changeDebitInterest(double newInterest) throws BankException {
        if (newInterest <= 0) throw new BankException("Interest must be positive!");
        debitInterest = newInterest;

        events.notify("debit interest");
    }

    public void changeCreditFee(double newFee) throws BankException {
        if (newFee <= 0) throw new BankException("Fee must be positive
!");
        creditFee = newFee;

        events.notify("credit fee");
    }

    public void changeCreditLimit(double newLimit) throws
BankException {
        if (newLimit <= 0) throw new BankException("Limit must be
positive!");
        creditFee = newLimit;

        events.notify("credit limit");
    }

    @Override
    public void update(String eventType) throws BankException {
        switch (eventType) {
            case "daily payment" -> calculateDailyPayment();
            case "monthly payment" -> payReward();
            default -> {
                var e = new IllegalArgumentException();
                throw new BankException("Invalid event!", e);
            }
        }
    }

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

        Bank other = (Bank) obj;

        return other.getName().equals(this.getName());
    }

    @Override
    public int hashCode() {
        return name.hashCode();
    }
}
```

Листинг 1.11: BankConsole.java

```java
package bank;

import java.util.HashMap;
import java.util.Scanner;

public class BankConsole {
    public BankDTO collectBankData() {
        Scanner in = new Scanner(System.in);

        System.out.println("Enter bank name:");
        String name = in.nextLine();

        System.out.println("Enter debit interest:");
        double debitInterest = in.nextDouble();

        System.out.println("Enter credit fee:");
        double creditFee = in.nextDouble();

        System.out.println("Enter credit limit:");
        double creditLimit = in.nextDouble();

        System.out.println("Enter limit for unverified clients:");
        double unverifiedLimit = in.nextDouble();

        System.out.println("Enter deposit default interest:");
        double depositDefaultInterest = in.nextDouble();

        System.out.println("Enter how many conditions will be");
        int n = in.nextInt();

        HashMap<Double, Double> conditions = new HashMap<>(n);

        for (int i = 0; i < n; i++) {
            System.out.println("Enter amount border:");
            double amountBorder = in.nextDouble();

            System.out.println("Enter interest for this border");
            double interestBorder = in.nextDouble();

            conditions.put(amountBorder, interestBorder);
        }

        return new BankDTO(name, debitInterest, creditFee, creditLimit
  ,
                unverifiedLimit, depositDefaultInterest, conditions);
    }
}
```

Листинг 1.12: BankDTO.java

```
package bank;

import java.util.Map;

public class BankDTO {

    private String name;
    private double debitInterest;
    private double creditFee;
    private double creditLimit;
    private double unverifiedLimit;
    private double depositDefaultInterest;
    private Map<Double, Double> depositInterestConditions;

    public BankDTO(String name, double debitInterest, double creditFee
    , double creditLimit,
                    double unverifiedLimit, double
    depositDefaultInterest, Map<Double, Double>
    depositInterestConditions) {
        this.name = name;
        this.debitInterest = debitInterest;
        this.creditFee = creditFee;
        this.creditLimit = creditLimit;
        this.unverifiedLimit = unverifiedLimit;
        this.depositDefaultInterest = depositDefaultInterest;
        this.depositInterestConditions = depositInterestConditions;
    }

    public double getCreditFee() {
        return creditFee;
    }

    public double getCreditLimit() {
        return creditLimit;
    }

    public double getDebitInterest() {
        return debitInterest;
    }

    public String getName() {
        return name;
    }

    public double getDepositDefaultInterest() {
        return depositDefaultInterest;
    }

    public double getUnverifiedLimit() {
        return unverifiedLimit;
    }

    public Map<Double, Double> getDepositInterestConditions() {
        return depositInterestConditions;
    }
}
```

Листинг 1.13: CentralBank.java

```java
package bank;

import account.IAccount;
import tools.BankException;
import tools.EventManager;

import java.time.Duration;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

public class CentralBank {

    private final List<Bank> banks;
    private final List<Transaction> transactions;

    public EventManager events;

    public CentralBank() {
        banks = new ArrayList<>();
        transactions = new ArrayList<>();

        events = new EventManager("daily payment", "monthly payment");
    }

    public List<Bank> getBanks() {
        return banks;
    }

    public List<Transaction> getTransactions() {
        return transactions;
    }

    public Bank addBank(BankDTO bankData) throws BankException {
        if (bankData == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Bank can not be null!", e);
        }

        for (Bank bank: banks) {
            if (Objects.equals(bank.getName(), bankData.getName())) {
                throw new BankException("Bank with such name already
   exist!");
            }
        }

        Bank bank = new Bank(this, bankData);
        banks.add(bank);

        return bank;
    }

    public Transaction makeTransaction(IAccount from, double amount,
    IAccount to) throws BankException {
        if (from == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Account can not be null!", e);
        }

        if (to == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Account can not be null!", e);
        }

        if (amount <= 0) throw new BankException("Amount can not be
    negative!");

        from.takeMoney(amount);
        to.addMoney(amount);

        var transaction = new Transaction(from, amount, to);
        transactions.add(transaction);

        return transaction;
    }

    public void cancelTransaction(Transaction transaction) throws
    BankException {
        if (transaction == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Transaction can not be null!", e)
    ;
        }

        transaction.getFrom().addMoney(transaction.getAmount());
        transaction.getTo().takeMoney(transaction.getAmount());

        transactions.remove(transaction);
    }

    public void calculateIncome(LocalDate from, LocalDate to) throws
    BankException {
        long daysBetween = Duration.between(from, to).toDays();
        for (long i = 0; i < daysBetween; i++) {
            events.notify("daily payment");
            if (i % 30 == 0 && i > 0) events.notify("monthly payment")
    ;
        }
        }


}
```

Листинг 1.14: Transaction.java

```java
package bank;

import account.IAccount;
import tools.BankException;

public class Transaction {
    private final double amount;
    private final IAccount from;
    private final IAccount to;

    public Transaction(IAccount from, double amount, IAccount to)
  throws BankException {
        if (from == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Account can not be null!", e);
        }

        if (to == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Account can not be null!", e);
        }

        if (amount <= 0) throw new BankException("Amount can not be
  negative!");

        this.amount = amount;
        this.from = from;
        this.to = to;
    }

    public double getAmount() {
        return amount;
    }

    public IAccount getFrom() {
        return from;
    }

    public IAccount getTo() {
        return to;
    }

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

        Transaction other = (Transaction) obj;

        return from.equals(other.from) && to.equals(other.to) &&
    amount == other.amount;
    }

    @Override
    public int hashCode() {
        return from.hashCode() + to.hashCode() - (int) amount;
    }
}
```

Листинг 1.15: Client.java

```java
package client;

import account.IAccount;
import bank.Bank;
import tools.BankException;
import tools.IEventListener;

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

public class Client implements IEventListener {

    private final String name;
    private final String surname;

    private String address;
    private String passport;

    private final UUID id;

    private boolean verified;

    private final List<IAccount> accounts;

    private final Bank bank;

    public Client(ClientDTO clientData, Bank bank) throws
  BankException {

        if (clientData == null){
            var e = new IllegalArgumentException();
            throw new BankException("Client data can not be null!", e)
  ;
        }

        if (bank == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Bank can not be null!", e);
        }

        this.bank = bank;

        if (clientData.getName() == null) throw new
  IllegalArgumentException();
        if (clientData.getSurname() == null) throw new
  IllegalArgumentException();

        name = clientData.getName();
        surname = clientData.getSurname();

        if (clientData.getAddress() == null) throw new IllegalArgumentException();
        if (clientData.getPassport() == null) throw new IllegalArgumentException();

        verified = !clientData.getAddress().equals("LATER") && !clientData.getPassport().equals("LATER");

        address = clientData.getAddress();
        passport = clientData.getPassport();

        id = clientData.getId();
        accounts = new ArrayList<>();
    }

    public String getName() {
        return name;
    }

    public String getSurname() {
        return surname;
    }

    public String getAddress() {
        return address;
    }

    public String getPassport() {
        return passport;
    }

    public UUID getId() {
        return id;
    }

    public boolean getVerified() {
        return verified;
    }

    public List<IAccount> getAccounts() {
        return accounts;
    }

    public void addMissingData(ClientDTO clientData) throws BankException {
        if (clientData == null){
            var e = new IllegalArgumentException();
            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
    public int hashCode() {
        return id.hashCode();
    }
}
```

Листинг 1.16: ClientConsole.java

```java
package client;

import tools.BankException;

import java.util.Scanner;

public class ClientConsole {
    public ClientDTO collectPersonalData() {
        Scanner in = new Scanner(System.in);

        System.out.println("Enter your name:");
        String name = in.nextLine();

        System.out.println("Enter your surname:");
        String surname = in.nextLine();

        System.out.println("Enter your address:");
        String address = in.nextLine();

        System.out.println("Enter your passport:");
        String passport = in.nextLine();

        return new ClientDTO(name, surname, address, passport);
    }

    public ClientDTO addMissingData(ClientDTO clientData) throws
    BankException {
        if (clientData == null) {
            var e = new IllegalArgumentException();
            throw new BankException("Client data can not be null!", e)
    ;
        }

        String address = "LATER";
        String passport = "LATER";
        Scanner in = new Scanner(System.in);

        if (clientData.getAddress().equals("LATER")) {
            System.out.println("Enter your address^");
            address = in.nextLine();
        }

        if (clientData.getPassport().equals("LATER")) {
            System.out.println("Enter your passport");
            passport = in.nextLine();
        }

        return new ClientDTO(clientData.getName(), clientData.
    getSurname(), address, passport);
```

```
        }
}
```

Листинг 1.17: ClientDTO.java

```java
package client;

import java.util.UUID;

public class ClientDTO {

    private final String name;
    private final String surname;
    private final String address;
    private final String passport;
    private final UUID id;

    public ClientDTO(String name, String surname, String address,
  String passport) {
        this.name = name;
        this.surname = surname;
        this.address = address;
        this.passport = passport;

        id = UUID.randomUUID();
    }

    public String getName() {
        return name;
    }

    public String getSurname() {
        return surname;
    }

    public String getAddress() {
        return address;
    }

    public String getPassport() {
        return passport;
    }

    public UUID getId() {
        return id;
    }
}
```

Листинг 1.18: BankException.java

```
package tools;

public class BankException extends Exception {

    public BankException() {
        super();
    }

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

    public BankException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

Листинг 1.19: EventManager.java

```
package tools;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

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

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

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

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

    public void notify(String eventType) throws BankException {
        List<IEventListener> users = listeners.get(eventType);
        for (IEventListener listener : users) {
            listener.update(eventType);
        }
    }
}
```

Листинг 1.20: IEventListener.java

```java
package tools;

public interface IEventListener {
    void update(String eventType) throws BankException;
}
```

Листинг 1.21: ControllerException.java

```
package controller.tools;

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

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

    public ControllerException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

Листинг 1.22: Color.java

```java
package dao.colors;

public enum Color {
    Black,
    White,
    Gray,
    Orange
}
```

Листинг 1.23: Cat.java

```
package dao.entities;

import dao.colors.Color;

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

@Entity
@Table(name = "cats", schema = "public", catalog = "postgres")
public class Cat {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Basic
    @Column(name = "id")
    private long id;
    @Basic
    @Column(name = "name")
    private String name;
    @Basic
    @Column(name = "birthdate")
    private Timestamp birthdate;
    @Basic
    @Column(name = "species")
    private String species;
    @Basic
    @Column(name = "color")
    @Enumerated(EnumType.STRING)
    private Color 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 getBirthdate() {
        return birthdate;
    }
```

```java

    public void setBirthdate(Timestamp birthdate) {
        this.birthdate = birthdate;
    }

    public String getSpecies() {
        return species;
    }

    public void setSpecies(String species) {
        this.species = species;
    }

    public Object getColor() {
        return color;
    }

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

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

    @Override
    public int hashCode() {
        return Objects.hash(id, name, birthdate, species, color);
    }

    @Override
    public String toString() {
        return id + ", " + name + ", " + birthdate + ", " + species + ", " + color;
    }
}
```

Листинг 1.24: CatsFriends.java

```
package dao.entities;

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

@Entity
@Table(name = "catsfriends", schema = "public", catalog = "postgres")
public class CatsFriends {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id")
    private long id;
    @Basic
    @Column(name = "first_cat_id")
    private long firstCatId;
    @Basic
    @Column(name = "second_cat_id")
    private long secondCatId;

    public long getId() {
        return id;
    }

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

    public long getFirstCatId() {
        return firstCatId;
    }

    public void setFirstCatId(long firstCatId) {
        this.firstCatId = firstCatId;
    }

    public long getSecondCatId() {
        return secondCatId;
    }

    public void setSecondCatId(long secondCatId) {
        this.secondCatId = secondCatId;
    }

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

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

Листинг 1.25: Owner.java

```java
package dao.entities;

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

@Entity
@Table(name = "owner", schema = "public", catalog = "postgres")
public class Owner {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id")
    private long id;
    @Basic
    @Column(name = "name")
    private String name;
    @Basic
    @Column(name = "birthdate")
    private Timestamp birthdate;

    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 getBirthdate() {
        return birthdate;
    }

    public void setBirthdate(Timestamp birthdate) {
        this.birthdate = birthdate;
    }

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

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

    @Override
    public String toString() {
        return id + ", " + name + ", " + birthdate;
    }
}
```

Листинг 1.26: OwnersCats.java

```
package dao.entities;

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

@Entity
@Table(name = "ownerscats", schema = "public", catalog = "postgres")
public class OwnersCats {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id")
    private long id;
    @Basic
    @Column(name = "owner_id")
    private long ownerId;
    @Basic
    @Column(name = "cat_id")
    private long catId;

    public long getId() {
        return id;
    }

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

    public long getOwnerId() {
        return ownerId;
    }

    public void setOwnerId(long ownerId) {
        this.ownerId = ownerId;
    }

    public long getCatId() {
        return catId;
    }

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

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

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

Листинг 1.27: CatDAO.java

```
package dao.implementations;

import dao.interfaces.DAO;
import dao.entities.Cat;
import dao.tools.DAOException;
import dao.tools.HibernateSessionUtil;
import org.hibernate.HibernateException;
import org.hibernate.Session;

import java.util.List;

public class CatDAO implements DAO<Cat> {
    @Override
    public List<Cat> getAll() {
        List<Cat> entities;

        Session session = HibernateSessionUtil.getSessionFactory().
    openSession();

        session.getTransaction().begin();
        entities = session.createQuery(
                "select cats from Cat cats", Cat.class).getResultList
    ();
        session.getTransaction().commit();

        session.close();

        return entities;
    }

    @Override
    public void add(Cat cat) throws DAOException {
        try {
            Session session = HibernateSessionUtil.getSessionFactory()
    .openSession();

            session.getTransaction().begin();
            session.save(cat);
            session.getTransaction().commit();

            session.close();

        } catch (HibernateException e) {
            throw new DAOException(e.getMessage(), e);
        }
    }

    @Override
    public Cat getById(Long id) throws DAOException {
        Cat cat;
        try {
            Session session = HibernateSessionUtil.getSessionFactory()
    .openSession();

            session.getTransaction().begin();
            cat = session.get(Cat.class, id);
            session.getTransaction().commit();

            session.close();

        } catch (HibernateException e) {
            throw new DAOException(e.getMessage(), e);
        }

        return cat;
    }

    @Override
    public void update(Cat cat) throws DAOException {
        try {
            Session session = HibernateSessionUtil.getSessionFactory()
    .openSession();

            session.getTransaction().begin();
            session.update(cat);
            session.getTransaction().commit();

            session.close();

        } catch (HibernateException e) {
            throw new DAOException(e.getMessage(), e);
        }
    }

    @Override
    public void delete(Cat cat) throws DAOException {
        try {
            Session session = HibernateSessionUtil.getSessionFactory()
    .openSession();

            session.getTransaction().begin();
            session.delete(cat);
            session.getTransaction().commit();

            session.close();

        } catch (HibernateException e) {
            throw new DAOException(e.getMessage(), e);
        }
        }
}
```

Листинг 1.28: CatsFriendsDAO.java

```java
package dao.implementations;

import dao.entities.CatsFriends;
import dao.interfaces.DAO;
import dao.tools.DAOException;
import dao.tools.HibernateSessionUtil;
import org.hibernate.HibernateException;
import org.hibernate.Session;

import java.util.List;

public class CatsFriendsDAO implements DAO<CatsFriends> {
    @Override
    public List<CatsFriends> getAll() {
        List<CatsFriends> entities;

        Session session = HibernateSessionUtil.getSessionFactory().
    openSession();

        session.getTransaction().begin();
        entities = session.createQuery(
                "select catsfrineds from CatsFriends catsfrineds",
    CatsFriends.class).getResultList();
        session.getTransaction().commit();

        session.close();

        return entities;
    }

    @Override
    public void add(CatsFriends catsFriends) throws DAOException {
        try {
            Session session = HibernateSessionUtil.getSessionFactory()
    .openSession();

            session.getTransaction().begin();
            session.save(catsFriends);
            session.getTransaction().commit();

            session.close();

        } catch (HibernateException e) {
            throw new DAOException(e.getMessage(), e);
        }
    }

    @Override
    public CatsFriends getById(Long id) throws DAOException {
        CatsFriends catsFriends;
        try {
            Session session = HibernateSessionUtil.getSessionFactory()
    .openSession();

            session.getTransaction().begin();
            catsFriends = session.get(CatsFriends.class, id);
            session.getTransaction().commit();

            session.close();

        } catch (HibernateException e) {
            throw new DAOException(e.getMessage());
        }

        return catsFriends;
    }

    @Override
    public void update(CatsFriends catsFriends) throws DAOException {
        try {
            Session session = HibernateSessionUtil.getSessionFactory()
    .openSession();

            session.getTransaction().begin();
            session.update(catsFriends);
            session.getTransaction().commit();

            session.close();

        } catch (HibernateException e) {
            throw new DAOException(e.getMessage(), e);
        }
    }

    @Override
    public void delete(CatsFriends catsFriends) throws DAOException {
        try {
            Session session = HibernateSessionUtil.getSessionFactory()
    .openSession();

            session.getTransaction().begin();
            session.delete(catsFriends);
            session.getTransaction().commit();

            session.close();

        } catch (HibernateException e) {
            throw new DAOException(e.getMessage(), e);
        }
        }
}
```

Листинг 1.29: OwnerDAO.java

```
package dao.implementations;

import dao.interfaces.DAO;
import dao.entities.Owner;
import dao.tools.DAOException;
import dao.tools.HibernateSessionUtil;
import org.hibernate.HibernateException;
import org.hibernate.Session;

import java.util.List;

public class OwnerDAO implements DAO<Owner> {
    @Override
    public List<Owner> getAll() {
        List<Owner> entities;

        Session session = HibernateSessionUtil.getSessionFactory().
    openSession();

        session.getTransaction().begin();
        entities = session.createQuery(
                "select owners from Owner owners", Owner.class).
    getResultList();
        session.getTransaction().commit();

        session.close();

        return entities;
    }

    @Override
    public void add(Owner owner) throws DAOException {
        try {
            Session session = HibernateSessionUtil.getSessionFactory()
    .openSession();

            session.getTransaction().begin();
            session.save(owner);
            session.getTransaction().commit();

            session.close();

        } catch (HibernateException e) {
            throw new DAOException(e.getMessage());
        }
    }

    @Override
    public Owner getById(Long id) throws DAOException {
        Owner owner;
        try {
            Session session = HibernateSessionUtil.getSessionFactory()
    .openSession();

            session.getTransaction().begin();
            owner = session.get(Owner.class, id);
            session.getTransaction().commit();

            session.close();

        } catch (HibernateException e) {
            throw new DAOException(e.getMessage());
        }

        return owner;
    }

    @Override
    public void update(Owner owner) throws DAOException {
        try {
            Session session = HibernateSessionUtil.getSessionFactory()
    .openSession();

            session.getTransaction().begin();
            session.update(owner);
            session.getTransaction().commit();

            session.close();

        } catch (HibernateException e) {
            throw new DAOException(e.getMessage());
        }
    }

    @Override
    public void delete(Owner owner) throws DAOException {
        try {
            Session session = HibernateSessionUtil.getSessionFactory()
    .openSession();

            session.getTransaction().begin();
            session.delete(owner);
            session.getTransaction().commit();

            session.close();

        } catch (HibernateException e) {
            throw new DAOException(e.getMessage());
        }
        }
}
```

Листинг 1.30: OwnersCatsDAO.java

```
package dao.implementations;

import dao.entities.OwnersCats;
import dao.interfaces.DAO;
import dao.tools.DAOException;
import dao.tools.HibernateSessionUtil;
import org.hibernate.HibernateException;
import org.hibernate.Session;

import java.util.List;

public class OwnersCatsDAO implements DAO<OwnersCats> {
    @Override
    public List<OwnersCats> getAll() {
        List<OwnersCats> entities;

        Session session = HibernateSessionUtil.getSessionFactory().
openSession();

        session.getTransaction().begin();
        entities = session.createQuery(
                "select ownerscats from OwnersCats ownerscats",
OwnersCats.class).getResultList();
        session.getTransaction().commit();

        session.close();

        return entities;
    }

    @Override
    public void add(OwnersCats ownersCats) throws DAOException {
        try {
            Session session = HibernateSessionUtil.getSessionFactory()
.openSession();

            session.getTransaction().begin();
            session.save(ownersCats);
            session.getTransaction().commit();

            session.close();

        } catch (HibernateException e) {
            throw new DAOException(e.getMessage(), e);
        }
    }

    @Override
    public OwnersCats getById(Long id) throws DAOException {
        OwnersCats ownersCats;
        try {
            Session session = HibernateSessionUtil.getSessionFactory()
    .openSession();

            session.getTransaction().begin();
            ownersCats = session.get(OwnersCats.class, id);
            session.getTransaction().commit();

            session.close();

        } catch (HibernateException e) {
            throw new DAOException(e.getMessage());
        }

        return ownersCats;
    }

    @Override
    public void update(OwnersCats ownersCats) throws DAOException {
        try {
            Session session = HibernateSessionUtil.getSessionFactory()
    .openSession();

            session.getTransaction().begin();
            session.update(ownersCats);
            session.getTransaction().commit();

            session.close();

        } catch (HibernateException e) {
            throw new DAOException(e.getMessage(), e);
        }
    }

    @Override
    public void delete(OwnersCats ownersCats) throws DAOException {
        try {
            Session session = HibernateSessionUtil.getSessionFactory()
    .openSession();

            session.getTransaction().begin();
            session.delete(ownersCats);
            session.getTransaction().commit();

            session.close();

        } catch (HibernateException e) {
            throw new DAOException(e.getMessage(), e);
        }
        }
}
```

Листинг 1.31: DAO.java

```java
package dao.interfaces;

import dao.tools.DAOException;

import java.util.List;

public interface DAO<T> {
    List<T> getAll();
    void add(T t) throws DAOException;
   T getById(Long id) throws DAOException;
    void update(T t) throws DAOException;
    void delete(T t) throws DAOException;
}
```

Листинг 1.32: 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 1.33: HibernateSessionUtil.java

```java
package dao.tools;

import org.hibernate.SessionFactory;

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

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

    private static SessionFactory initSessionFactory() {
        try {
            return new Configuration().configure().buildSessionFactory
  ();
        } catch (Throwable e) {
            throw new ExceptionInInitializerError(e);
        }
    }

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

        return sessionFactory;
    }
}
```

Листинг 1.34: Service.java

```java
package service;

import dao.colors.Color;
import dao.entities.Cat;
import dao.entities.CatsFriends;
import dao.entities.Owner;
import dao.entities.OwnersCats;
import dao.implementations.CatDAO;
import dao.implementations.CatsFriendsDAO;
import dao.implementations.OwnerDAO;
import dao.implementations.OwnersCatsDAO;
import dao.tools.DAOException;
import service.tools.ServiceException;

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

public class Service {
    private CatDAO catDAO;
    private OwnerDAO ownerDAO;
    private CatsFriendsDAO catsFriendsDAO;
    private OwnersCatsDAO ownersCatsDAO;

    public Service(CatDAO catDAO, OwnerDAO ownerDAO, CatsFriendsDAO
  catsFriendsDAO, OwnersCatsDAO ownersCatsDAO) {
        this.catDAO = catDAO;
        this.ownerDAO = ownerDAO;
        this.catsFriendsDAO = catsFriendsDAO;
        this.ownersCatsDAO = ownersCatsDAO;
    }

    public Cat addCat(String name, Timestamp birthdate, String species
  , Color color) throws ServiceException {
        Cat cat = new Cat();

        cat.setName(name);
        cat.setBirthdate(birthdate);
        cat.setSpecies(species);
        cat.setColor(color);

        try {
            catDAO.add(cat);
        } catch (DAOException e) {
            throw new ServiceException("Problems with data access" + e
  .getMessage());
        }

        return cat;
    }

    public void removeCat(Cat cat) throws ServiceException {
        if (cat == null) {
            throw new ServiceException("Cat entity can not be null!");
        }

        List<CatsFriends> catsAndFriends = catsFriendsDAO.getAll();

        for (CatsFriends catAndFriend: catsAndFriends) {
            if (catAndFriend.getFirstCatId() == cat.getId() ||
catAndFriend.getSecondCatId() == cat.getId()) {
                try {
                    catsFriendsDAO.delete(catAndFriend);
                } catch (DAOException e) {
                    throw new ServiceException("Problem with data
access" + e.getMessage());
                }
            }
        }

        List<OwnersCats> ownersAndCats = ownersCatsDAO.getAll();

        for (OwnersCats ownerAndCat: ownersAndCats) {
            if (ownerAndCat.getCatId() == cat.getId()) {
                try {
                    ownersCatsDAO.delete(ownerAndCat);
                } catch (DAOException e) {
                    throw new ServiceException("Problem with data
access" + e.getMessage());
                }
            }
        }

        try {
            catDAO.delete(cat);
        } catch (DAOException e) {
            throw new ServiceException("Problems with data access" + e
.getMessage());
        }
    }

    public Owner addOwner(String name, Timestamp birthdate) throws
ServiceException {
        Owner owner = new Owner();

        owner.setName(name);
        owner.setBirthdate(birthdate);

        try {
            ownerDAO.add(owner);
```

```java
        } catch (DAOException e) {
            throw new ServiceException("Problems with data access" + e.getMessage());
        }

        return owner;
    }

    public void removeOwner(Owner owner) throws ServiceException {
        if (owner == null) {
            throw new ServiceException("Owner entity can not be null!");
        }

        List<OwnersCats> ownersAndCats = ownersCatsDAO.getAll();

        for (OwnersCats ownerAndCat: ownersAndCats) {
            if (ownerAndCat.getOwnerId() == owner.getId()) {
                try {
                    ownersCatsDAO.delete(ownerAndCat);
                } catch (DAOException e) {
                    throw new ServiceException("Problem with data access" + e.getMessage());
                }
            }
        }

        try {
            ownerDAO.delete(owner);
        } catch (DAOException e) {
            throw new ServiceException("Problems with data access" + e.getMessage());
        }
    }

    public OwnersCats addCatToOwner(Cat cat, Owner owner) throws ServiceException {
        if (cat == null) {
            throw new ServiceException("Cat entity can not be null!");
        }

        if (owner == null) {
            throw new ServiceException("Owner entity can not be null!");
        }

        List<OwnersCats> ownersAndCats = ownersCatsDAO.getAll();

        for (OwnersCats ownerAndCat: ownersAndCats) {
            if (ownerAndCat.getOwnerId() == owner.getId() &&
ownerAndCat.getCatId() == cat.getId()) {
                throw new ServiceException("Such owner-cat pair
already exist!");
            }
        }

        OwnersCats ownersCats = new OwnersCats();

        ownersCats.setOwnerId(owner.getId());
        ownersCats.setCatId(cat.getId());

        try {
            ownersCatsDAO.add(ownersCats);
        } catch (DAOException e) {
            throw new ServiceException("Problems with data access" + e
.getMessage());
        }

        return ownersCats;
    }

    public void removeCatFromOwner(Cat cat, Owner owner) throws
ServiceException {
        if (cat == null) {
            throw new ServiceException("Cat entity can not be null!");
        }

        if (owner == null) {
            throw new ServiceException("Owner entity can not be null!"
);
        }

        List<OwnersCats> ownersAndCats = ownersCatsDAO.getAll();

        for (OwnersCats ownerAndCat: ownersAndCats) {
            if (ownerAndCat.getOwnerId() == owner.getId() &&
ownerAndCat.getCatId() == cat.getId()) {
                try {
                    ownersCatsDAO.delete(ownerAndCat);
                } catch (DAOException e) {
                    throw new ServiceException("Problems with data
access" + e.getMessage());
                }
            }
        }
    }

    public CatsFriends makeFriends(Cat firstCat, Cat secondCat) throws
    ServiceException {
        if (firstCat == null) {
            throw new ServiceException("Cat entity can not be null!");
        }

        if (secondCat == null) {
            throw new ServiceException("Cat entity can not be null!");
        }

        CatsFriends catsFriends = new CatsFriends();

        catsFriends.setFirstCatId(firstCat.getId());
        catsFriends.setSecondCatId(secondCat.getId());

        try {
            catsFriendsDAO.add(catsFriends);
        } catch (DAOException e) {
            throw new ServiceException("Problems with data access" + e
    .getMessage());
        }

        return catsFriends;
    }

    public void breakFriendship(Cat firstCat, Cat secondCat) throws
    ServiceException {
        if (firstCat == null) {
            throw new ServiceException("Cat entity can not be null!");
        }

        if (secondCat == null) {
            throw new ServiceException("Cat entity can not be null!");
        }

        List<CatsFriends> catsAndFriends = catsFriendsDAO.getAll();

        for (CatsFriends catAndFriend: catsAndFriends) {
            if (catAndFriend.getFirstCatId() == firstCat.getId() &&
    catAndFriend.getSecondCatId() == secondCat.getId() ||
                    catAndFriend.getFirstCatId() == secondCat.getId()
    && catAndFriend.getSecondCatId() == firstCat.getId()) {
                try {
                    catsFriendsDAO.delete(catAndFriend);
                } catch (DAOException e) {
                    throw new ServiceException("Problems with data
    access" + e.getMessage());
                }
            }
        }
    }

    public List<Cat> getAllCats() {
        return catDAO.getAll();
    }

    public List<Owner> getAllOwners() {
        return ownerDAO.getAll();
    }

}
```

Листинг 1.35: ServiceException.java

```java
package service.tools;

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

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

    public ServiceException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

Листинг 1.36: ServiceTest.java

```java
package service;

import dao.colors.Color;
import dao.entities.Cat;
import dao.entities.CatsFriends;
import dao.entities.Owner;
import dao.entities.OwnersCats;
import dao.implementations.CatDAO;
import dao.implementations.CatsFriendsDAO;
import dao.implementations.OwnerDAO;
import dao.implementations.OwnersCatsDAO;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import org.mockito.junit.jupiter.MockitoExtension;

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

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

@ExtendWith(MockitoExtension.class)
class ServiceTest {

    @Mock
    private CatDAO catDAO;
    @Mock
    private OwnerDAO ownerDAO;
    @Mock
    private CatsFriendsDAO catsFriendsDAO;
    @Mock
    private OwnersCatsDAO ownersCatsDAO;

    private Service service;

    public ServiceTest() {
        MockitoAnnotations.openMocks(this);
        this.service = new Service(catDAO, ownerDAO, catsFriendsDAO,
   ownersCatsDAO);
    }

    @Test
    void addCat_Should_Return_True() throws Exception {
        Cat cat = new Cat();

        cat.setName("default");
        cat.setBirthdate(Timestamp.valueOf("2012-12-12 00:00:00.000000000"));
        cat.setSpecies("default");
        cat.setColor(Color.White);

        given(catDAO.getById(1L)).willReturn(cat);

        Cat returnedCat = service.addCat("default", Timestamp.valueOf("2012-12-12 00:00:00.000000000"),
                "default", Color.White);

        assertEquals(catDAO.getById(1L), returnedCat);
    }

    @Test
    void addOwner_Should_Return_True() throws Exception {
        Owner owner = new Owner();

        owner.setName("default");
        owner.setBirthdate(Timestamp.valueOf("2012-12-12 00:00:00.000000000"));

        given(ownerDAO.getById(1L)).willReturn(owner);

        Owner returnedOwner = service.addOwner("default", Timestamp.valueOf("2012-12-12 00:00:00.000000000"));

        assertEquals(ownerDAO.getById(1L), returnedOwner);
    }

    @Test
    void addCatToOwner_Should_Return_True() throws Exception {
        OwnersCats ownersCats = new OwnersCats();

        ownersCats.setCatId(1L);
        ownersCats.setOwnerId(1L);

        given(ownersCatsDAO.getById(1L)).willReturn(ownersCats);

        Cat cat = new Cat();

        cat.setId(1L);
        cat.setName("default");
        cat.setBirthdate(Timestamp.valueOf("2012-12-12 00:00:00.000000000"));
        cat.setSpecies("default");
        cat.setColor(Color.White);

        Owner owner = new Owner();

        owner.setId(1L);
        owner.setName("default");
        owner.setBirthdate(Timestamp.valueOf("2012-12-12
    00:00:00.000000000"));

        OwnersCats returnedOwnersCats = service.addCatToOwner(cat,
    owner);

        assertEquals(ownersCatsDAO.getById(1L).getCatId(),
    returnedOwnersCats.getCatId());
        assertEquals(ownersCatsDAO.getById(1L).getOwnerId(),
    returnedOwnersCats.getOwnerId());

    }

    @Test
    void makeFriends_Should_Return_True() throws Exception {
        CatsFriends catsFriends = new CatsFriends();

        catsFriends.setFirstCatId(1L);
        catsFriends.setSecondCatId(2L);

        given(catsFriendsDAO.getById(1L)).willReturn(catsFriends);

        Cat cat = new Cat();

        cat.setId(1L);
        cat.setName("default");
        cat.setBirthdate(Timestamp.valueOf("2012-12-12
    00:00:00.000000000"));
        cat.setSpecies("default");
        cat.setColor(Color.White);

        Cat cat1 = new Cat();

        cat1.setId(2L);
        cat1.setName("default");
        cat1.setBirthdate(Timestamp.valueOf("2012-12-12
    00:00:00.000000000"));
        cat1.setSpecies("default");
        cat1.setColor(Color.White);

        CatsFriends returnedCatsFriends = service.makeFriends(cat,
    cat1);

        assertEquals(catsFriendsDAO.getById(1L).getFirstCatId(),
    returnedCatsFriends.getFirstCatId());
        assertEquals(catsFriendsDAO.getById(1L).getSecondCatId(),
    returnedCatsFriends.getSecondCatId());
    }

    @Test
    void getAllCats_Should_Return_True() {
        given(catDAO.getAll()).willReturn(new ArrayList<>());

        List<Cat> returnedList = service.getAllCats();

        assertEquals(catDAO.getAll(), returnedList);
    }

    @Test
    void getAllOwners_Should_Return_True() {
        given(ownerDAO.getAll()).willReturn(new ArrayList<>());

        List<Owner> returnedList = service.getAllOwners();

        assertEquals(ownerDAO.getAll(), returnedList);
    }
}
```