**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский университет ИТМО»**

Факультет информационных технологий и программирования

Лабораторная работа № 2

*Сервис по учету котиков и их владельцев*

**Выполнил студент группы M32051**

Писарева Юлия Игоревна

****

**Подпись:**

**Проверил:**

Чикишев Константин Максимович

Санкт-Петербург

2022

**Текст задания**

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.

**Решение с комментариями**

package entities;

import jakarta.persistence.\*;

import models.Color;

import java.io.Serializable;

import java.time.LocalDateTime;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.List;

@Entity

@Table(name = "cats")

public class Cat implements Serializable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@Column(name = "birth\_date")

private LocalDateTime birthDate;

private String breed;

private Color color;

@ManyToOne

@JoinColumn(name = "owner\_id")

private Owner owner;

@ManyToMany

@JoinTable(

name = "cat\_friends",

joinColumns = { @JoinColumn(name = "left\_id") },

inverseJoinColumns = { @JoinColumn(name = "right\_id") }

)

private List<Cat> friends;

public Cat(String name, LocalDateTime birthDate, String breed, Color color, Owner owner) {

this.name = name;

this.birthDate = birthDate;

this.breed = breed;

this.color = color;

this.owner = owner;

this.friends = new ArrayList<>();

}

public Cat(String name, LocalDateTime birthDate, String breed, Color color) {

this(name, birthDate, breed, color, null);

}

protected Cat() { }

public Long getId() { return id; }

public String getName() { return name; }

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

public LocalDateTime getBirthDate() { return birthDate; }

public String getBreed() { return breed; }

public Color getColor() { return color; }

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

public Owner getOwner() { return owner; }

public void setOwner(Owner owner) { this.owner = owner; }

public List<Cat> getFriends() { return Collections.unmodifiableList(friends); }

public void addFriend(Cat... cats) { Collections.addAll(friends, cats); }

public void removeFriend(Cat... cats) { friends.removeAll(Arrays.asList(cats)); }

}

package entities;

import jakarta.persistence.\*;

import java.io.Serializable;

import java.time.LocalDateTime;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.List;

@Entity

@Table(name = "owners")

public class Owner implements Serializable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@Column(name = "birth\_date")

private LocalDateTime birthDate;

@OneToMany(mappedBy = "owner")

private List<Cat> cats;

public Owner(String name, LocalDateTime birthDate) {

this.name = name;

this.birthDate = birthDate;

this.cats = new ArrayList<>();

}

protected Owner() { }

public Long getId() { return id; }

public String getName() { return name; }

public LocalDateTime getBirthDate() { return birthDate; }

public List<Cat> getCats() { return Collections.unmodifiableList(cats); }

public void addCat(Cat... cats) { Collections.addAll(this.cats, cats); }

public void removeCat(Cat... cats) { this.cats.removeAll(Arrays.asList(cats)); }

}

package models;

public enum Color {

WHITE,

BLACK,

ORANGE,

GRAY,

BLUE,

}

package services;

import entities.Cat;

import models.Color;

public class CatService {

public void renameCat(Cat cat, String name) { cat.setName(name); }

void friendCats(Cat left, Cat right) {

left.addFriend(right);

right.addFriend(left);

}

void unfriendCats(Cat left, Cat right) {

left.removeFriend(right);

right.removeFriend(left);

}

void paintCat(Cat cat, Color color) { cat.setColor(color); }

}

package services;

import entities.Cat;

import entities.Owner;

public class OwnerService {

public void addCat(Owner owner, Cat cat) {

if (cat.getOwner() != null) {

removeCat(cat.getOwner(), cat);

}

owner.addCat(cat);

cat.setOwner(owner);

}

public void removeCat(Owner owner, Cat cat) {

owner.removeCat(cat);

cat.setOwner(null);

}

}

package daos;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import java.io.Serializable;

import java.util.List;

import java.util.function.Consumer;

import java.util.function.Function;

public abstract class AbstractDao<T extends Serializable> {

private Class<T> entityType;

protected SessionFactory sessionFactory;

public AbstractDao(Class<T> entityType, SessionFactory sessionFactory) {

this.entityType = entityType;

this.sessionFactory = sessionFactory;

}

public void setEntityType(final Class<T> entityType) {

this.entityType = entityType;

}

public T findOne(final long id) {

return executeTransactionWithValue(s -> s.get(entityType, id));

}

public List<T> findAll() {

return executeTransactionWithValue(s -> s

.createQuery("from " + entityType.getName(), entityType)

.list());

}

public T create(final T entity) {

executeTransaction(s -> s.persist(entity));

return entity;

}

public T update(final T entity) { return executeTransactionWithValue(s -> s.merge(entity)); }

public void delete(final T entity) { executeTransaction(s -> s.remove(entity)); }

public void deleteById(final long entityId) {

executeTransaction(s -> {

T entity = findOne(entityId);

delete(entity);

});

}

protected void executeTransaction(Consumer<Session> action) {

var session = sessionFactory.getCurrentSession();

var transaction = session.beginTransaction();

action.accept(session);

transaction.commit();

}

protected <R> R executeTransactionWithValue(Function<Session, R> function) {

var session = sessionFactory.getCurrentSession();

var transaction = session.beginTransaction();

var value = function.apply(session);

transaction.commit();

return value;

}

}

package daos;

import entities.Cat;

import org.hibernate.SessionFactory;

public class CatDao extends AbstractDao<Cat> {

public CatDao(Class<Cat> entityType, SessionFactory sessionFactory) { super(entityType, sessionFactory); }

}

package daos;

import entities.Owner;

import org.hibernate.SessionFactory;

public class OwnerDao extends AbstractDao<Owner> {

public OwnerDao(Class<Owner> entityType, SessionFactory sessionFactory) { super(entityType, sessionFactory); }

}

package tools;

public class KotikiExeption extends RuntimeException {

private Integer errorCode;

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

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

public Integer getErrorCode() { return errorCode; }

}

package tools;

import entities.Cat;

import entities.Owner;

import org.hibernate.SessionFactory;

import org.hibernate.boot.Metadata;

import org.hibernate.boot.MetadataSources;

import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

import org.hibernate.service.ServiceRegistry;

import java.util.HashMap;

import java.util.Map;

public class SessionFactoryBuilder {

public SessionFactory build(String url, String username, String password, String dialect, String driverClass) {

Map<String, Object> settings = new HashMap<>();

settings.put("connection.driver\_class", driverClass);

settings.put("dialect", "org.hibernate.dialect." + dialect);

settings.put("hibernate.connection.url", url);

settings.put("hibernate.connection.username", username);

settings.put("hibernate.connection.password", password);

settings.put("hibernate.current\_session\_context\_class", "thread");

settings.put("hibernate.show\_sql", "true");

settings.put("hibernate.format\_sql", "true");

ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder()

.applySettings(settings).build();

MetadataSources metadataSources = new MetadataSources(serviceRegistry);

metadataSources.addAnnotatedClass(Owner.class);

metadataSources.addAnnotatedClass(Cat.class);

Metadata metadata = metadataSources.buildMetadata();

return metadata.getSessionFactoryBuilder().build();

}

}

package services;

import daos.CatDao;

import daos.OwnerDao;

import entities.Cat;

import entities.Owner;

public class InfrastructureOwnerService {

private final OwnerService ownerService;

private final CatDao catDao;

private final OwnerDao ownerDao;

public InfrastructureOwnerService(OwnerService ownerService, CatDao catDao, OwnerDao ownerDao) {

this.ownerService = ownerService;

this.catDao = catDao;

this.ownerDao = ownerDao;

}

public void addCat(Owner owner, Cat cat) {

ownerService.addCat(owner, cat);

ownerDao.update(owner);

catDao.update(cat);

}

public void removeCat(Owner owner, Cat cat) {

ownerService.removeCat(owner, cat);

ownerDao.update(owner);

catDao.update(cat);

}

public Owner addToDatabase(Owner owner) { return ownerDao.create(owner); }

}

package services;

import daos.CatDao;

import entities.Cat;

import models.Color;

public class InfrastrutureCatService {

private final CatService catService;

private final CatDao catDao;

public InfrastrutureCatService(CatService catService, CatDao catDao) {

this.catService = catService;

this.catDao = catDao;

}

public void renameCat(Cat cat, String name) {

catService.renameCat(cat, name);

catDao.update(cat);

}

public void friendCats(Cat left, Cat right) {

catService.friendCats(left, right);

catDao.update(left);

catDao.update(right);

}

public void unfriendCats(Cat left, Cat right) {

catService.unfriendCats(left, right);

catDao.update(left);

catDao.update(right);

}

public void paintCat(Cat cat, Color color) {

catService.paintCat(cat, color);

catDao.update(cat);

}

public Cat addToDatabase(Cat cat) { return catDao.create(cat); }

}

import daos.CatDao;

import daos.OwnerDao;

import entities.Cat;

import entities.Owner;

import models.Color;

import services.CatService;

import services.InfrastructureOwnerService;

import services.InfrastrutureCatService;

import services.OwnerService;

import tools.SessionFactoryBuilder;

import java.time.LocalDateTime;

public class Main {

public static void main(String[] args) {

try {

var url = args[0];

var username = args[1];

var password = args[2];

var dialect = args[3];

var driver = args[4];

var sessionFactoryBuilder = new SessionFactoryBuilder();

var sessionFactory = sessionFactoryBuilder

.build(url, username, password, dialect, driver);

var catService = new CatService();

var ownerService = new OwnerService();

var catDao = new CatDao(Cat.class, sessionFactory);

var ownerDao = new OwnerDao(Owner.class, sessionFactory);

var infrastructureCatService = new InfrastrutureCatService(catService, catDao);

var infrastructureOwnerService = new InfrastructureOwnerService(

ownerService, catDao, ownerDao);

var date = LocalDateTime.of(2018, 12, 12, 0, 0, 0);

var catOne = new Cat("Барсик", date, "Бродяга", Color.WHITE);

var catTwo = new Cat("Сизый", date, "Симпотяга", Color.BLUE);

var owner = new Owner("Кошачий барон", date);

infrastructureOwnerService.addToDatabase(owner);

infrastructureCatService.addToDatabase(catOne);

infrastructureCatService.addToDatabase(catTwo);

infrastructureCatService.friendCats(catOne, catTwo);

infrastructureOwnerService.addCat(owner, catOne);

} catch (Exception e) {

System.out.println();

}

}

}

import daos.CatDao;

import daos.OwnerDao;

import entities.Cat;

import entities.Owner;

import models.Color;

import org.hibernate.SessionFactory;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestInstance;

import services.CatService;

import services.InfrastructureOwnerService;

import services.InfrastrutureCatService;

import org.mockito.\*;

import services.OwnerService;

import java.time.LocalDateTime;

import static org.junit.jupiter.api.Assertions.assertTrue;

import static org.mockito.Matchers.any;

import static org.mockito.Mockito.\*;

@TestInstance(TestInstance.Lifecycle.PER\_CLASS)

public class KotikiTest {

private InfrastructureOwnerService infrastructureOwnerService;

private InfrastrutureCatService infrastrutureCatService;

private CatDao catDao;

private OwnerDao ownerDao;

@BeforeEach

public void setup() {

catDao = Mockito.mock(CatDao.class);

when(catDao.update(any(Cat.class))).thenAnswer(i -> i.getArguments()[0]);

ownerDao = Mockito.mock(OwnerDao.class);

when(ownerDao.update(any(Owner.class))).thenAnswer(i -> i.getArguments()[0]);

infrastructureOwnerService = new InfrastructureOwnerService(new OwnerService(), catDao, ownerDao);

infrastrutureCatService = new InfrastrutureCatService(new CatService(), catDao);

}

@Test

public void addCatToOwner\_OwnerCatsContainsCat() {

LocalDateTime date1 = LocalDateTime.of(2001, 12, 12, 11, 0, 3);

LocalDateTime date2 = LocalDateTime.of(2020, 1, 3, 4, 12, 0);

Owner owner = new Owner("Виктор", date1);

Cat cat = new Cat("Федя", date2, "Дворняга", Color.ORANGE);

infrastructureOwnerService.addCat(owner, cat);

verify(ownerDao, times(1)).update(owner);

verify(catDao, times(1)).update(cat);

assertTrue(owner.getCats().contains(cat));

}

@Test

public void makeCatsFriends\_BothCatsUpdated() {

LocalDateTime date1 = LocalDateTime.of(2019, 12, 12, 11, 0, 3);

LocalDateTime date2 = LocalDateTime.of(2020, 1, 3, 4, 12, 0);

Cat cat1 = new Cat("Виктор", date1, "Дворняга", Color.BLACK);

Cat cat2 = new Cat("Федя", date2, "Дворняга", Color.ORANGE);

infrastrutureCatService.friendCats(cat1, cat2);

verify(catDao, times(1)).update(cat1);

verify(catDao, times(1)).update(cat2);

assertTrue(cat1.getFriends().contains(cat2));

}

}