|  |
| --- |
|  |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |
| Институт информационных технологий |
| Кафедра инструментального и прикладного программного обеспечения |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКИМ РАБОТАМ** | |
| **по дисциплине** | |
| **«Шаблоны программных платформ языка Джава»** | |
|  | |
| Выполнил студент группы ИКБО-03-19 | Прокофьев С. Я. |
| Принял ассистент кафедры ИиППО | Ермаков С.Р. |

|  |  |  |
| --- | --- | --- |
| Практические работы выполнены | «\_\_»\_\_\_\_\_\_\_2021г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_2021г. | *(подпись руководителя)* |

**Оглавление**

[Практическая работа №1 3](#_Toc69499650)

[Практическая работа №2 4](#_Toc69499651)

[Практическая работа №3 7](#_Toc69499652)

[Практическая работа №4 12](#_Toc69499653)

[Практическая работа №5 13](#_Toc69499654)

[Практическая работа №6 14](#_Toc69499655)

[Практическая работа №7 19](#_Toc69499656)

[Практическая работа №8 22](#_Toc69499657)

[Практическая работа №9 25](#_Toc69499658)

[Практическая работа №10 27](#_Toc69499659)

[Практическая работа №11 29](#_Toc69499660)

[Практическая работа №12 31](#_Toc69499661)

[Практическая работа №13 33](#_Toc69499662)

[Практическая работа №14 35](#_Toc69499663)

[Практическая работа №15 39](#_Toc69499664)

[Практическая работа №16 44](#_Toc69499665)

[Практическая работа №17 49](#_Toc69499666)

[Практическая работа №18 54](#_Toc69499667)

[Практическая работа №19 57](#_Toc69499668)

[Практическая работа №20 60](#_Toc69499669)

[Практическая работа №21 63](#_Toc69499670)

[Практическая работа №22 67](#_Toc69499671)

[Практическая работа №23 69](#_Toc69499672)

[Практическая работа №24 82](#_Toc69499673)

# Практическая работа №1

**Задание:** Имплементировать интерфейс Function, получающий на вход пару чисел и возвращающий наибольший общий делитель.

**Цель работы:** Знакомство со встроенными функциональными интерфейсами Java. Возможности Java 8. Лямбда-выражения. Области действия, замыкания. Предикаты. Функции. Компараторы.

**Выполнение работы:**

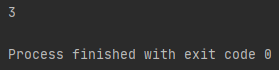
**Function.java**

package prac\_1;  
  
@FunctionalInterface  
public interface Function<T, R>{  
 R apply(T t);  
}

**Task\_1.java**

package prac\_1;  
  
public class Task\_1 {  
 public static void main(String[] args) {  
 int[] arr = {15, 24};  
 Function<int[], String> ans = x -> euclid(arr);  
 System.out.println(ans.apply(arr));  
 }  
 public static String euclid(int[] arr){  
 while (arr[1] !=0) {  
 int tmp = arr[0] % arr[1];  
 arr[0] = arr[1];  
 arr[1] = tmp;  
 }  
 return String.valueOf(arr[0]);  
 }  
}

**Результаты:**

****

**Выводы:**

В результате проделанной работы познакомились со встроенными функциональными интерфейсами Java и лямбда-выражениями.

# Практическая работа №2

**Задание:** В ходе выполнения практической работы должно быть реализовано:  
1) класс Human (int age, String firstName, String lastName, LocalDate birthDate, int weight);  
2) приложение, которое создает список из объектов класса Human, а затем производит действия в соответствии с вариантом индивидуального задания (список после каждого этапа должен выводиться в консоль).  
Все действия должны производиться только с использованием Stream API.  
Индивидуальное задание должно быть оформлено в отдельном проекте.  
Для проверки работоспособности выполненного индивидуального задания следует использовать отдельный класс с методом main.

**Вариант:** Сортировка по второй букве имени, фильтрация по весу кратно 10, сортировка по произведению веса на возраст, произведение всех весов.

**Цель работы:** Работа со Stream API в Java 8.

**Выполнение работы:**

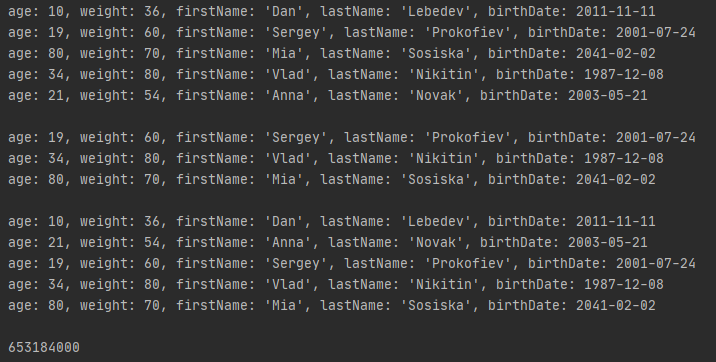
**Human.java**

package prac\_2;  
  
import java.time.LocalDate;  
  
public class Human {  
 private final int age, weight;  
 private final String firstName, lastName;  
 private final LocalDate birthDate;  
 Human(int age, String firstName, String lastName, LocalDate birthDate, int weight){  
 this.age = age;  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.birthDate = birthDate;  
 this.weight = weight;  
 }  
 public String getSecondLetterOfFirstName(){  
 return firstName.substring(1);  
 }  
  
 public int getWeight() {  
 return weight;  
 }  
  
 public int getMultiplication(){  
 return weight\*age;  
 }  
  
 @Override  
 public String toString() {  
 return "age: " + age +  
 ", weight: " + weight +  
 ", firstName: '" + firstName + '\'' +  
 ", lastName: '" + lastName + '\'' +  
 ", birthDate: " + birthDate;  
 }  
}

**Task\_2.java**

package prac\_2;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.Comparator;  
import java.util.stream.Stream;  
  
public class Task\_2 {  
 public static void main(String[] args){  
 ArrayList<Human> arr = new ArrayList<>();  
 arr.add(new Human(19, "Sergey", "Prokofiev",  
 LocalDate.of(2001, 7, 24), 60));  
 arr.add(new Human(21, "Anna", "Novak",  
 LocalDate.of(2003, 5, 21), 54));  
 arr.add(new Human(34, "Vlad", "Nikitin",  
 LocalDate.of(1987, 12, 8), 80));  
 arr.add(new Human(80, "Mia", "Sosiska",  
 LocalDate.of(2041, 2, 2), 70));  
 arr.add(new Human(10, "Dan", "Lebedev",  
 LocalDate.of(2011, 11, 11), 36));  
 Stream<Human> stream1 = arr.stream();  
 stream1.sorted(Comparator.comparing(Human::getSecondLetterOfFirstName))  
 .forEach(System.out::println);  
 System.out.println();  
 Stream<Human> stream2 = arr.stream();  
 stream2.filter(human -> human.getWeight()%10 == 0)  
 .forEach(System.out::println);  
 System.out.println();  
 Stream<Human> stream3 = arr.stream();  
 stream3.sorted(Comparator.comparingInt(Human::getMultiplication))  
 .forEach(System.out::println);  
 int ans = arr.stream()  
 .mapToInt(Human::getWeight)  
 .reduce(1, (a,b) -> a\*b);  
 System.out.println();  
 System.out.println(ans);  
 }  
}

**Результаты:**

****

**Выводы:**

Проделав данную работу, мы изучили новый способ взаимодействия с коллекциями объектов в языке Java, называемый Stream API, и преобразовали данную коллекцию к нужному виду.

# Практическая работа №3

**Задание:** Создать свои потокобезопасные имплементации интерфейсов в соответствии с вариантом индивидуального задания.

**Вариант:** Set с использованием ключевого слова synchronized, Map с использованием Lock.

**Цель работы:** Знакомство с конкурентным программированием в Java. Потокобезопасность, ключевое слово syncrhonized, мьютексы, семафоры, мониторы, барьеры.

**Выполнение работы:**

**MyMap.java**

package prac\_3;  
  
import java.util.Collection;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.Set;  
import java.util.concurrent.locks.Lock;  
import java.util.concurrent.locks.ReentrantLock;  
  
  
public class MyMap<K, V> extends HashMap<K, V> implements Map<K, V> {  
 private static final Lock lock = new ReentrantLock();  
 @Override  
 public int size() {  
 lock.lock();  
 int c = super.size();  
 lock.unlock();  
 return c;  
 }  
  
 @Override  
 public boolean isEmpty() {  
 lock.lock();  
 boolean c = super.isEmpty();  
 lock.unlock();  
 return c;  
 }  
  
 @Override  
 public boolean containsKey(Object key) {  
 lock.lock();  
 boolean c = super.containsKey(key);  
 lock.unlock();  
 return c;  
 }  
  
 @Override  
 public boolean containsValue(Object value) {  
 lock.lock();  
 boolean c = super.containsValue(value);  
 lock.unlock();  
 return c;  
 }  
  
 @Override  
 public V get(Object key) {  
 lock.lock();  
 V c = super.get(key);  
 lock.unlock();  
 return c;  
 }  
  
 @Override  
 public V put(K key, V value) {  
 lock.lock();  
 V c = super.put(key, value);  
 lock.unlock();  
 return c;  
 }  
  
 @Override  
 public V remove(Object key) {  
 lock.lock();  
 V c = super.remove(key);  
 lock.unlock();  
 return c;  
 }  
  
 @Override  
 public void putAll(Map<? extends K, ? extends V> m) {  
 lock.lock();  
 super.putAll(m);  
 lock.unlock();  
 }  
  
 @Override  
 public void clear() {  
 lock.lock();  
 super.clear();  
 lock.unlock();  
 }  
  
 @Override  
 public Set<K> keySet() {  
 lock.lock();  
 Set<K> c = super.keySet();  
 lock.unlock();  
 return c;  
 }  
  
 @Override  
 public Collection<V> values() {  
 lock.lock();  
 Collection<V> c = super.values();  
 lock.unlock();  
 return c;  
 }  
  
 @Override  
 public Set<Entry<K, V>> entrySet() {  
 lock.lock();  
 Set<Entry<K, V>> c = super.entrySet();  
 lock.unlock();  
 return c;  
 }  
}

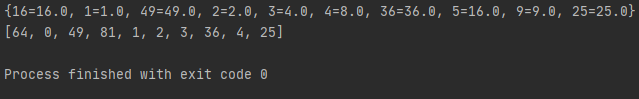
**MySet.java**

package prac\_3;  
  
import java.util.Collection;  
import java.util.HashSet;  
import java.util.Iterator;  
import java.util.Set;  
  
public class MySet<E> extends HashSet<E> implements Set<E> {  
 @Override  
 synchronized public int size() {  
 return super.size();  
 }  
  
 @Override  
 synchronized public boolean isEmpty() {  
 return super.isEmpty();  
 }  
  
 @Override  
 synchronized public boolean contains(Object o) {  
 return super.contains(o);  
 }  
  
 @Override  
 synchronized public Iterator<E> iterator() {  
 return super.iterator();  
 }  
  
 @Override  
 synchronized public Object[] toArray() {  
 return super.toArray();  
 }  
  
 @Override  
 synchronized public <T> T[] toArray(T[] a) {  
 return super.toArray(a);  
 }  
  
 @Override  
 synchronized public boolean add(E e) {  
 return super.add(e);  
 }  
  
 @Override  
 synchronized public boolean remove(Object o) {  
 return super.remove(o);  
 }  
  
 @Override  
 synchronized public boolean containsAll(Collection<?> c) {  
 return super.containsAll(c);  
 }  
  
 @Override  
 synchronized public boolean addAll(Collection<? extends E> c) {  
 return super.addAll(c);  
 }  
  
 @Override  
 synchronized public boolean retainAll(Collection<?> c) {  
 return super.retainAll(c);  
 }  
  
 @Override  
 synchronized public boolean removeAll(Collection<?> c) {  
 return super.removeAll(c);  
 }  
  
 @Override  
 synchronized public void clear() {  
 super.clear();  
 }  
}

**Task\_3.java**

package prac\_3;  
  
public class Task\_3 {  
 public static void main(String[] args) throws InterruptedException {  
 MyMap<Integer, Double> map = new MyMap<>();  
 MySet<Integer> set = new MySet<>();  
 Thread one = new Thread(()->{  
 for (int i = 0; i < 5; i++) {  
 map.put(i+1, Math.pow(2, i));  
 }  
 });  
 Thread two = new Thread(()->{  
 for (int i = 3; i < 8; i++) {  
 map.put(i\*i, Math.pow(i, 2));  
 }  
 });  
 Thread third = new Thread(()->{  
 for (int i = 0; i < 5; i++) {  
 set.add(i);  
 }  
 });  
 Thread four = new Thread(()->{  
 for (int i = 5; i < 10; i++) {  
 set.add(i\*i);  
 }  
 });  
 one.start();  
 two.start();  
 third.start();  
 four.start();  
 Thread.sleep(3000);  
 System.out.println(map);  
 System.out.println(set);  
 }  
}

**Результаты:**

****

**Выводы:** В ходе данной практической работы освоена работа с созданием потокобезопасных классов, использованием ключевого слова syncronized и классов Semaphore, Lock в Java.

# Практическая работа №4

**Задание:** Реализовать собственную имплементацию ExecutorService с единственным параметром конструктора – количеством потоков.

**Цель работы:** Работа с ExecutorService, CompletableFuture.

**Выполнение работы:**

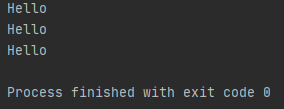
**MyExecutorService.java**

package prac\_4;  
  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
  
public class MyExecutorService {  
 MyExecutorService(int num){  
 ExecutorService e = Executors.newFixedThreadPool(num);  
 for(int i = 0; i < num; i++) {  
 e.submit(() -> System.out.println("Hello"));  
 }  
 e.shutdown();  
 }  
}

**Task\_4.java**

package prac\_4;  
  
public class Task\_4 {  
 public static void main(String[] args) {  
 MyExecutorService e = new MyExecutorService(3);  
 }  
}

**Результаты:**



**Выводы:** В ходе данной практической работы мы познакомились с работой с потоками, используя абстракции - "ExecutorsService" и "CompletableFuture" в Java. Узнали, что такое асинхронность и попробовали такие реализации как: callback-функции, async/await, корутины, реактивность.

# Практическая работа №5

**Задание:** Реализовать паттерн Singleton как минимум 3-мя способами.

**Цель работы:** Познакомиться с паттернами проектирования, их определением и классификацией. Обзор паттернов GoF. Паттерн Синглтон.

**Выполнение работы:**

**Singleton1.java**

package prac\_5;  
  
public enum Singleton1 {  
 INSTANCE;  
 public static Singleton1 getInstance() {  
 return INSTANCE;  
 }  
  
}

**Singleton2.java**

package prac\_5;  
  
public class Singleton2 {  
 private static Singleton2 instance;  
 private Singleton2(){}  
 public static synchronized Singleton2 getInstance() {  
 if(instance == null) {  
 instance = new Singleton2();  
 }  
 return instance;  
 }  
  
}

**Singleton3.java**

package prac\_5;  
  
public class Singleton3 {  
 private Singleton3() {}  
 private static class Singleton {  
 static final Singleton3 INSTANCE = new Singleton3();  
 }  
 public static Singleton3 getInstance() {  
 return Singleton.INSTANCE;  
 }  
}

**Выводы:** Проделав данную работу, мы разработали несколько реализаций паттерна Singletone на языке Java.

# Практическая работа №6

**Задание:** Написать реализацию паттернов «Фабричный метод», «Абстрактная фабрика», «Строитель», «Прототип».

**Цель работы:** Знакомство с реализацией порождающих паттернов проектирования.

**Выполнение работы:**

*Абстрактная фабрика:*

**A\_Test.java**

package prac\_6.abstract\_factory;  
  
public class A\_Test {  
 public static void main(String[] args) {  
 PeopleFactory pf = new PeopleFactory();  
 Man man = pf.createMan();  
 Woman woman = pf.createWoman();  
 man.FixSomething();  
 woman.CookSomething();  
 }  
}

**Chef.java**

package prac\_6.abstract\_factory;  
  
public class Chef implements Woman{  
 @Override  
 public void CookSomething() {  
 System.out.println("Done");  
 }  
}

**Man.java**

package prac\_6.abstract\_factory;  
  
public interface Man {  
 void FixSomething();  
}

**People.java**

package prac\_6.abstract\_factory;  
  
public abstract class People {  
 public abstract Woman createWoman();  
 public abstract Man createMan();  
}

**PeopleFactory.java**

package prac\_6.abstract\_factory;  
  
public class PeopleFactory extends People{  
 @Override  
 public Woman createWoman() {  
 return new Chef();  
 }  
  
 @Override  
 public Man createMan() {  
 return new Plumber();  
 }  
}

**Plumber.java**

package prac\_6.abstract\_factory;  
  
public class Plumber implements Man{  
 @Override  
 public void FixSomething() {  
 System.out.println("Done");  
 }  
}

**Woman.java**

package prac\_6.abstract\_factory;  
  
public interface Woman {  
 void CookSomething();  
}

*Строитель:*

**Builder.java**

package prac\_6.builder;  
  
public interface Builder {  
 void CreateComplex();  
 void BuildPart1(int part1);  
 void BuildPart2(int part2);  
}

**B\_Test.java**

package prac\_6.builder;  
  
public class B\_Test {  
 public static void main(String[] args) {  
 ConcreteBuilder B = new ConcreteBuilder();  
 Director D = new Director();  
 D.Construct(B);  
 Complex C = B.GetResult();  
 System.out.println("C.re = " + C.re);  
 System.out.println("C.im = " + C.im);  
 }  
}

**Complex.java**

package prac\_6.builder;  
  
public class Complex {  
 public double re;  
 public double im;  
}

**ConcreteBuilder.java**

package prac\_6.builder;  
  
public class ConcreteBuilder implements Builder{  
 private Complex currentBuilder;  
 @Override  
 public void CreateComplex() {  
 currentBuilder = new Complex();  
 }  
  
 @Override  
 public void BuildPart1(int part1) {  
 currentBuilder.re = part1;  
 }  
  
 @Override  
 public void BuildPart2(int part2) {  
 currentBuilder.im = part2;  
 }  
 Complex GetResult() {  
 return currentBuilder;  
 }  
}

**Director.java**

package prac\_6.builder;  
  
public class Director {  
 void Construct(Builder builder) {  
 builder.CreateComplex();  
 builder.BuildPart1(15);  
 builder.BuildPart2(30);  
 }  
}

*Фабричный метод:*

**F\_Test.java**

package prac\_6.factory\_method;  
  
public class F\_Test {  
 public static void main(String[] args) {  
 HumanFactory hf = new HumanFactory();  
 Human st = hf.getHuman("Student");  
 st.sleep();  
 Human sc = hf.getHuman("Schoolboy");  
 sc.sleep();  
 }  
}

**Human.java**

package prac\_6.factory\_method;  
  
public interface Human {  
 void sleep();  
}

**HumanFactory.java**

package prac\_6.factory\_method;  
  
public class HumanFactory {  
 public Human getHuman(String human){  
 if (human.equals("Student")){  
 return new Student();  
 }  
 else if(human.equals("Schoolboy")){  
 return new Schoolboy();  
 }  
 return null;  
 }  
}

**Schoolboy.java**

package prac\_6.factory\_method;  
  
public class Schoolboy implements Human{  
 @Override  
 public void sleep() {  
 System.out.println("No");  
 }  
}

**Student.java**

package prac\_6.factory\_method;  
  
public class Student implements Human{  
 @Override  
 public void sleep() {  
 System.out.println("Sleep");  
 }  
}

*Прототип:*

**Human.java**

package prac\_6.prototype;  
  
  
public class Human implements Prototype{  
 int age;  
 public Human(int age){  
 this.age = age;  
 }  
 @Override  
 public Human clone() {  
 return new Human(age);  
 }  
 public int getAge() {  
 return age;  
 }  
}

**Prototype.java**

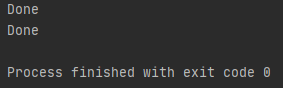
package prac\_6.prototype;  
  
public interface Prototype {  
 Prototype clone();  
}

**P\_Test.java**

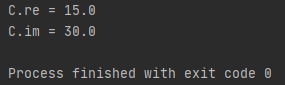
package prac\_6.prototype;  
  
public class P\_Test {  
 public static void main(String[] args) {  
 Human human = new Human(12);  
 Human kid = human.clone();  
 System.out.println(kid.getAge());  
 }  
}

**Результаты:**

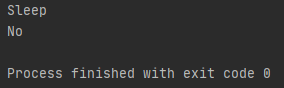
*Абстрактная фабрика:*

**

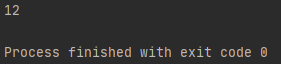
*Строитель:*

**

*Фабричный метод:*

**

*Прототип:*

**

**Выводы:**

В ходе данной работы были освоены на практике реализации паттернов «Фабричный метод», «Абстрактная фабрика», «Строитель», «Прототип» на языке Java.

# Практическая работа №7

**Задание:** Написать реализацию паттерна в соответствии с вариантом индивидуального задания.

**Вариант:** Декоратор, Фасад.

**Цель работы:** Реализация структурных паттернов проектирования.

**Выполнение работы:**

*Декоратор:*

**Car.java**

package prac\_7.decorator;  
  
public interface Car {  
 int getSpeed();  
 int getBaggageWeight();  
}

**D\_Test.java**

package prac\_7.decorator;  
  
public class D\_Test {  
 public static void main(String[] args) {  
 Car simpleCar = new SimpleCar();  
 System.out.println("Speed of simple car - " + String.valueOf(simpleCar.getSpeed()));  
 System.out.println("Simple car can accept baggage with weight " + String.valueOf(simpleCar.getBaggageWeight()));  
 Car sportCar = new SportCar(simpleCar);  
 System.out.println("Speed of sport car - " + String.valueOf(sportCar.getSpeed()));  
 System.out.println("Sport car can accept baggage with weight " + String.valueOf(sportCar.getBaggageWeight()));  
 }  
}

**SimpleCar.java**

package prac\_7.decorator;  
  
public class SimpleCar implements Car{  
 private int speed = 50;  
 private int baggageWeight = 100;  
  
 @Override  
 public int getSpeed() {  
 return this.speed;  
 }  
  
 @Override  
 public int getBaggageWeight() {  
 return this.baggageWeight;  
 }  
}

**SportCar.java**

package prac\_7.decorator;  
  
public class SportCar implements Car {  
 private final Car car;  
 public SportCar(Car car){  
 this.car = car;  
 }  
  
 @Override  
 public int getSpeed() {  
 return this.car.getSpeed() + 50;  
 }  
  
 @Override  
 public int getBaggageWeight() {  
 return this.car.getBaggageWeight();  
 }  
}

*Фасад:*

**Competition.java**

package prac\_7.facade;  
  
public class Competition {  
 private final Referee referee;  
 private final Sportsman sportsman;  
 private final Viewer viewer;  
  
 public Competition() {  
 this.referee = new Referee();  
 this.sportsman = new Sportsman();  
 this.viewer = new Viewer();  
 }  
  
 public void startCompetition() {  
 referee.GetTheWhistle();  
 sportsman.GetReady();  
 viewer.TakeFood();  
 }  
}

**F\_Test.java**

package prac\_7.facade;  
  
public class F\_Test {  
 public static void main(String[] args) {  
 Competition competition = new Competition();  
 competition.startCompetition();  
 }  
}

**Referee.java**

package prac\_7.facade;  
  
public class Referee {  
 public void GetTheWhistle(){  
 System.out.println("Судья готов");  
 }  
}

**Sportsman.java**

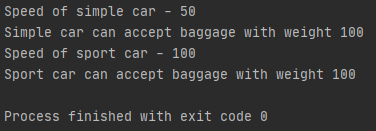
package prac\_7.facade;  
  
public class Sportsman {  
 public void GetReady(){  
 System.out.println("Спортсмен готов");  
 }  
}

**Viewer.java**

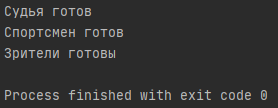
package prac\_7.facade;  
  
public class Viewer {  
 public void TakeFood(){  
 System.out.println("Зрители готовы");  
 }  
}

**Результаты:**

*Декоратор:*

**

*Фасад:*

****

**Выводы:**

В результате проделанной работы изучили структурные паттерны проектирования на Java.

# Практическая работа №8

**Задание:** Написать реализацию паттерна в соответствии с вариантом индивидуального задания.

**Вариант:** Посредник, Снимок.

**Цель работы:** Реализация поведенческих паттернов проектирования.

**Выполнение работы:**

*Посредник:*

**Colleague.java**

package prac\_8.mediator;  
  
public abstract class Colleague {  
 protected Mediator mediator;  
 public Colleague(Mediator mediator) {  
 this.mediator = mediator;  
 }  
 public void send(String message) {  
 mediator.send(message, this);  
 }  
 public abstract void notify(String message);  
}

**ConcreteColleague.java**

package prac\_8.mediator;  
  
public class ConcreteColleague extends Colleague{  
 public ConcreteColleague(Mediator mediator) {  
 super(mediator);  
 }  
  
 @Override  
 public void notify(String message) {  
 System.out.println("Коллега говорит: " + message);  
 }  
}

**ConcreteMediator.java**

package prac\_8.mediator;  
  
public class ConcreteMediator implements Mediator {  
 private ConcreteColleague colleague;  
  
 public void setColleague(ConcreteColleague colleague) {  
 this.colleague = colleague;  
 }  
  
 @Override  
 public void send(String message, Colleague sender) {  
 colleague.notify(message);  
 }  
}

**Mediator.java**

package prac\_8.mediator;  
  
public interface Mediator {  
 void send(String message, Colleague sender);  
}

*Снимок:*

**M\_Test.java**

package prac\_8.mediator;  
  
public class M\_Test {  
 public static void main(String[] args) {  
 ConcreteMediator m = new ConcreteMediator();  
 ConcreteColleague c = new ConcreteColleague(m);  
 m.setColleague(c);  
 c.send("Привет");  
 }  
}

**Caretaker.java**

package prac\_8.snapshot;  
  
public class Caretaker {  
 private Memento memento;  
 public Memento getMemento() {  
 return memento;  
 }  
 public void setMemento(Memento memento) {  
 this.memento = memento;  
 }  
}

**Memento.java**

package prac\_8.snapshot;  
  
public class Memento {  
 private final String state;  
 public Memento(String state) {  
 this.state = state;  
 }  
 public String getState() {  
 return state;  
 }  
}

**Originator.java**

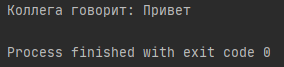
package prac\_8.snapshot;  
  
public class Originator {  
 private String state;  
 public void setState(String state) {  
 this.state = state;  
 }  
 public String getState() {  
 return state;  
 }  
 public Memento saveState() {  
 return new Memento(state);  
 }  
 public void restoreState(Memento memento) {  
 this.state = memento.getState();  
 }  
}

**S\_Test.java**

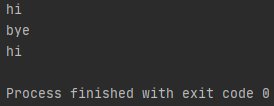
package prac\_8.snapshot;  
  
public class S\_Test {  
 public static void main(String[] args) {  
 Originator originator = new Originator();  
 Caretaker caretaker = new Caretaker();  
  
 originator.setState("hi");  
 System.out.println(originator.getState());  
 caretaker.setMemento(originator.saveState());  
  
 originator.setState("bye");  
 System.out.println(originator.getState());  
  
 originator.restoreState(caretaker.getMemento());  
 System.out.println(originator.getState());  
 }  
}

**Результаты:**

*Посредник:*

**

*Снимок:*

****

**Выводы:**

Проделав данную работу, мы изучили на практике поведенческие паттерны проектирования в Java.

# Практическая работа №9

**Задание:** Создать приложение, которое выводит какое-то сообщение в консоль. Создать Gradle Task, который создает jar-файл приложения, переносит его в отдельную папку, в которой хранится Dockerfile для jar, а затем создает Docker контейнер из данного jar-файла и запускает его.

**Цель работы:** Знакомство с системой сборки приложения. Gradle.

**Выполнение работы:**

**build.gradle**

plugins {  
 id 'java'  
}  
  
version '1.0'  
  
repositories {  
 mavenCentral()  
}  
  
  
task createJar(type: Jar) {  
 manifest {  
 attributes 'Main-Class': 'Main'  
 }  
 from { configurations.compile.collect { it.isDirectory() ? it : zipTree(it) } }  
 finalizedBy {  
 moveJar  
 }  
}  
  
task moveJar() {  
 doLast {  
 ant.move file: "build/libs/prac\_9-1.0.jar",  
 todir: "."  
 }  
 finalizedBy {  
 BuildDocker  
 }  
}  
  
task BuildDocker(type: Exec) {  
 workingDir projectDir  
 commandLine "docker", "build", "-t", "prac-9", "."  
 finalizedBy {  
 RunDocker  
 }  
}  
  
task RunDocker(type: Exec) {  
 commandLine "docker", "run", "-it", "-d", "--name", "serg", "prac-9"  
}

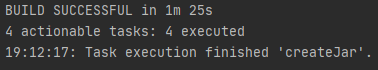
**Dockerfile**

FROM openjdk:15  
ENV JAR=prac\_9-1.0.jar  
COPY $JAR $JAR  
CMD java -jar $JAR

**Main.java**

package prac\_9;  
  
public class Main {  
 public static void main(String[] args) {  
 System.out.println("Hello Sergey");  
 }  
}

**Результаты:**

****

**Выводы:**

В результате проделанной работы научились создавать build.gradle, расписывать в нём задачи, зависимости и осуществлять через него сборку проекта и отдельных задач.

# Практическая работа №10

**Задание:** Создать приложение, в котором создается ApplicationContext и из него берётся бин с названием, переданным в качестве аргумента к приложению, и вызывается метод интерфейса, который он имплементирует. Нужно создать по одному бину для каждого класса, определить им название. Проверить, что вызывается при вводе названия каждого из бинов. Классы и интерфейс определяются в соответствии с вариантом индивидуального задания.

**Вариант:** Интерфейс Lighter с методом doLight(), его имплементации: Lamp, Flashlight, Firefly.

**Цель работы:** Введение в Spring. Container. Bean. Внедрение зависимостей, основанных на конструкторах и сеттерах. Конфигурация бинов. Автоматическое обнаружение и связывание классов.

**Выполнение работы:**

**BeanConfig.java**

import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class BeanConfig {  
 @Bean  
 public Firefly firefly(){  
 return new Firefly();  
 }  
 @Bean  
 public Flashlight flashlight(){  
 return new Flashlight();  
 }  
 @Bean  
 public Lamp lamp(){  
 return new Lamp();  
 }  
  
}

**Firefly.java**

public class Firefly implements Lighter{  
 @Override  
 public void doLight() {  
 System.out.println("Firefly");  
 }  
}

**Flashlight.java**

public class Flashlight implements Lighter{  
 @Override  
 public void doLight() {  
 System.out.println("Flashlight");  
 }  
}

**Lamp.java**

public class Lamp implements Lighter{  
 @Override  
 public void doLight() {  
 System.out.println("Lamp");  
 }  
}

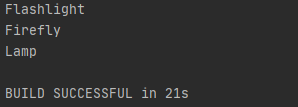
**Lighter.java**

public interface Lighter {  
 void doLight();  
}

**Test\_10.java**

import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
  
public class Test\_10 {  
 public static void main(String[] args) {  
 ApplicationContext context = new AnnotationConfigApplicationContext(BeanConfig.class);  
  
 Lighter lighter = context.getBean(Flashlight.class);  
 lighter.doLight();  
  
 lighter = context.getBean(Firefly.class);  
 lighter.doLight();  
  
 lighter = context.getBean(Lamp.class);  
 lighter.doLight();  
 }  
  
}

**Результаты:**

****

**Выводы:**

В результате проделанной работы научились использовать внедрение зависимостей и Bean с помощью Spring на языке Java.

# Практическая работа №11

**Задание:** Создать приложение с использованием Spring Boot Starter Initializr (https://start.spring.io/) с такими зависимостями:  
– Spring Web;  
– Lombok;  
– Validation;  
– Spring boot Actuator.  
Запустить приложение и удостовериться, что не появилось никаких ошибок. Добавить все эндпоинты в Actuator, сделать HTTP-запрос на проверку состояния приложения. Собрать jar-файл приложения, запустить и проверить состояние при помощи REST-запроса.

**Цель работы:** Разобраться с использованием Spring boot.

**Выполнение работы:**

**DemoApplication.java**

package sprboot.config;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class DemoApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.run(DemoApplication.class, args);  
 Test\_11 test11 = new Test\_11();  
 test11.test();  
 }  
}

**Test\_11.java**

package sprboot.config;  
  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class Test\_11 {  
 @RequestMapping("/Hello")  
 public String test(){  
 return "Hi";  
 }  
}

**HomeController.java**

package sprboot.controller;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.GetMapping;  
  
@Controller  
public class HomeController {  
  
 @GetMapping("/")  
 public String getIndex() {  
 return "index";  
 }  
}

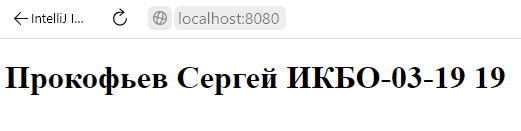
**application.properties**

management.endpoints.web.exposure.include=\*

**index.html**

<!DOCTYPE html>  
 <html>  
 <head>  
 <meta charset="UTF-8">  
 <title>Insert title here</title>  
 </head>  
 <body>  
 <h1>Прокофьев Сергей ИКБО-03-19 19</h1>  
 </body>  
</html>

**Результаты:**

**** 

**Выводы:** В результате проделанной работы разработано приложение с использованием Spring Boot Starter Initializr и необходимыми зависимостями.

# Практическая работа №12

**Задание:** Создать приложение, которое при запуске берет данные из одного файла, хеширует, а при остановке приложения удаляет исходный файл, оставляя только файл с захешированными данными. Названия первого и второго файла передаются в качестве аргументов при запуске. При отсутствии первого файла создает второй файл и записывает в него строку null. Реализовать с использованием аннотаций PostConstruct, PreDestroy.

**Цель работы:** Работа с жизненным циклом компонентов. Аннотации PostConstruct, PreDestroy.

**Выполнение работы:**

**DemoApplication.java**

package sprboot.config;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
  
@SpringBootApplication  
public class DemoApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.run(DemoApplication.class, args);  
 ApplicationContext applicationContext = new AnnotationConfigApplicationContext(Test\_12.class);  
 Test\_12 test = applicationContext.getBean(Test\_12.class);  
 }  
}

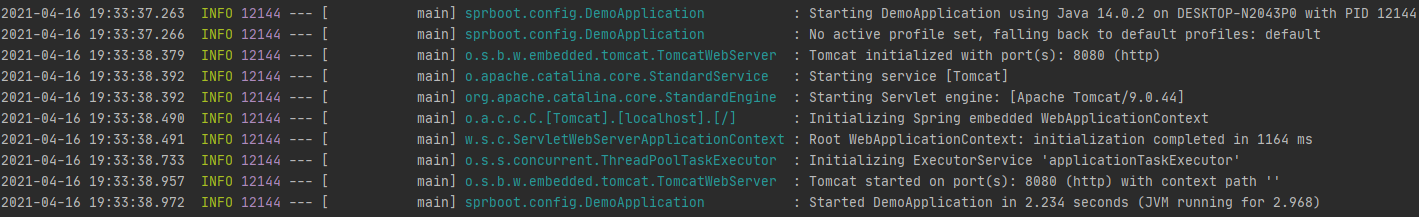
**Test\_12.java**

package sprboot.config;  
  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
import javax.annotation.PostConstruct;  
import javax.annotation.PreDestroy;  
import java.io.\*;  
  
@Component  
public class Test\_12 {  
 File file;  
 @Value("${file1}")  
 String file1;  
 @Value("${file2}")  
 String file2;  
 @PostConstruct  
 public void read() throws IOException {  
 file = new File(".\\Java\\sem\_4\\prac\_12\\src\\main\\resources", file1);  
 StringBuilder str = new StringBuilder();  
 if (!file.exists()) {  
 str = new StringBuilder("null");  
 }  
 else {  
 FileReader fr = new FileReader(file);  
 int i;  
 while ((i = fr.read()) != -1) {  
 str.append((char) i);  
 }  
 fr.close();  
 str = new StringBuilder(Integer.toString(str.toString().hashCode()));  
 }  
 File out = new File(".\\Java\\sem\_4\\prac\_12\\src\\main\\resources", file2);  
 if (out.exists()){  
 FileWriter fw = new FileWriter(out, false);  
 fw.write(str.toString());  
 fw.close();  
 }  
 }  
 @PreDestroy  
 public void delete(){  
 file = new File(".\\Java\\sem\_4\\prac\_12\\src\\main\\resources", file1);  
 if(file.delete()){  
 System.out.println("Папка была удалена");  
 } else System.out.println("Папка не была найдена");  
  
 }  
}

**application.properties**

file1=f1.txt  
file2=f2.txt

**Результаты:**

****

**Выводы:** В ходе данной практической работы мы научились использовать аннотации PostConstruct, PreDestroy и работать с жизненным циклом компонентов.

# Практическая работа №13

**Задание:** Создать файл application.yml в папке resources, добавить в него такие свойства:  
– student.name – имя студента;  
– student.last\_name – фамилия студента;  
– student.group – название группы студента.  
При запуске приложения выведите данные свойства в консоль при помощи интерфейса Environment или аннотации Value.

**Цель работы:** Конфигурирование приложения. Environment.

**Выполнение работы:**

**Student.java**

package config;  
  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
import javax.annotation.PostConstruct;  
  
  
@Component  
public class Student {  
 @Value("${student.name}")  
 String name;  
 @Value("${student.last\_name}")  
 String last\_name;  
 @Value("${student.group}")  
 String group;  
 @PostConstruct  
 public void init(){  
 System.out.println(name + " " + last\_name + " " + group);  
 }  
}

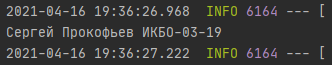
**Test\_13.java**

package config;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class Test\_13 {  
  
 public static void main(String[] args) {  
 SpringApplication.run(Test\_13.class, args);  
 }  
}

**application.yml**

student:  
 name: "Сергей"  
 last\_name: "Прокофьев"  
 group: "ИКБО-03-19"

**Результаты:**

****

**Выводы:** В результате проделанной работы научились конфигурировать приложения на Spring.

# Практическая работа №14

**Задание:** Создать отдельный репозиторий Git. Создать простой html-документ, который будет содержать вашу фамилию, имя, номер группы, номер варианта. Создать контроллер, который будет возвращать данный статический документ при переходе на url «/home». Выполнить задание в зависимости с вариантом индивидуального задания.

**Вариант:** Создать класс Departure с полями type,departureDate. Создать класс PostOffice с полями name, cityName. Создать классы-контроллеры для создания, удаления объектов и получения всех объектов каждого типа. Сами объекты хранить в памяти.

**Цель работы:** Знакомство со Spring MVC. Работа с Rest API в Spring.

**Выполнение работы:**

**Controllers.java**

package sprboot.config;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.ArrayList;  
import java.util.List;  
  
@Controller  
class HomeController {  
  
 @GetMapping("/home")  
 public String getIndex() {  
 return "Home.html";  
 }  
}  
@RequestMapping("/home/departure")  
@RestController  
class DepartureController {  
 List<Departure> list = new ArrayList<>();  
 @GetMapping("/add")  
 public String add(@RequestParam String type, @RequestParam String departureDate){  
 for (Departure departure : list) {  
 if(departure.departureDate.equals(departureDate)) {  
 return "На эту дату занято";  
 }  
 }  
 list.add(new Departure(type, departureDate));  
 return "Добавлен: " + type + " " + departureDate;  
 }  
 @GetMapping("/get")  
 public String get(){  
 StringBuilder str = new StringBuilder();  
 for (Departure departure : list) {  
 str.append(departure.getType()).append(" ").append(departure.getDepartureDate());  
 }  
 if (str.length() != 0)  
 return "Держи: " + str;  
 return "Пусто";  
 }  
 @GetMapping("/del/{departureDate}")  
 public String remove(@PathVariable String departureDate){  
 for(int i=0; i<list.size(); i++){  
 if(list.get(i).departureDate.equals(departureDate)){  
 String str = list.get(i).getType() + " " + list.get(i).getDepartureDate();  
 list.remove(i);  
 return "Удалил: " + str;  
 }  
 }  
 return "Такого нет";  
 }  
}  
@RestController  
class PostOfficeController {  
 List<PostOffice> list = new ArrayList<>();  
 @GetMapping("/add")  
 public String add(@RequestParam String name, @RequestParam String cityName){  
 for (PostOffice postOffice : list) {  
 if(postOffice.name.equals(name)) {  
 return "Такой уже есть";  
 }  
 }  
 list.add(new PostOffice(name, cityName));  
 return "Добавлен: " + name + " " + cityName;  
 }  
 @GetMapping("/get")  
 public String get(){  
 StringBuilder str = new StringBuilder();  
 for (PostOffice postOffice : list) {  
 str.append(postOffice.getName()).append(" ").append(postOffice.getCityName());  
 }  
 if (str.length() != 0)  
 return "Держи: " + str;  
 return "Пусто";  
 }  
 @GetMapping("/del/{name}")  
 public String remove(@PathVariable String name){  
 for(int i=0; i<list.size(); i++){  
 if(list.get(i).name.equals(name)){  
 String str = list.get(i).getName() + " " + list.get(i).getCityName();  
 list.remove(i);  
 return "Удалил: " + str;  
  
 }  
 }  
 return "Такого нет";  
 }  
}

**Departure.java**

package sprboot.config;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.Setter;  
  
@AllArgsConstructor  
@Getter  
@Setter  
public class Departure {  
 String type;  
 String departureDate;  
}

**PostOffice.java**

package sprboot.config;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.Setter;  
  
  
@AllArgsConstructor  
@Getter  
@Setter  
public class PostOffice {  
 String name;  
 String cityName;  
}

**Test14Application.java**

package sprboot.config;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class Test14Application {  
  
 public static void main(String[] args) {  
 SpringApplication.run(Test14Application.class, args);  
 }  
}

**Departure.html**

<!DOCTYPE html>  
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta charset="UTF-8">  
 <title>Insert title here</title>  
 </head>  
 <body>  
 <h1 th:text = "${type}"></h1>  
 <h1 th:text = "${departureDate}"></h1>  
 </body>  
</html>

**Home.html**

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8">  
 <title>Insert title here</title>  
 </head>  
 <body>  
 <h1>Прокофьев Сергей ИКБО-03-19 19</h1>  
 </body>  
</html>

**PostOffice.html**

<!DOCTYPE html>  
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta charset="UTF-8">  
 <title>Insert title here</title>  
 </head>  
 <body>  
 <h1 th:text = "${type}"></h1>  
 <h1 th:text = "${departureDate}"></h1>  
 </body>  
</html>

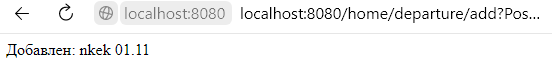
**Home.html**

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8">  
 <title>Insert title here</title>  
 </head>  
 <body>  
 <h1>Прокофьев Сергей ИКБО-03-19 19</h1>  
 </body>  
</html>

**index.html**

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
 </head>  
 <body>  
 <h1>Hello</h1>  
 </body>  
</html>

**Результаты:**

****

**Выводы:** Проделав данную работу, мы освоили знания в использовании Spring MVC и Rest API в Spring.

# Практическая работа №15

**Задание:** Изменить программу с предыдущего задания так, чтобы объекты хранились в базе данных PostgreSQL вместо памяти компьютера.

**Цель работы:** Использование Hibernate в Spring framework.

**Выполнение работы:**

**Config.java**

package prac15;  
  
import com.zaxxer.hikari.\*;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.orm.hibernate5.HibernateTransactionManager;  
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;  
import org.springframework.transaction.PlatformTransactionManager;  
  
import javax.sql.DataSource;  
import java.util.Properties;  
  
@Configuration  
public class Config {  
 @Bean  
 public HikariDataSource dataSource() {  
 HikariConfig config = new HikariConfig();  
 config.setDriverClassName("org.postgresql.Driver");  
 config.setJdbcUrl("jdbc:postgresql://localhost:5432/postgres");  
 config.setUsername("postgres");  
 config.setPassword("dratute");  
 return new HikariDataSource(config);  
 }  
  
 @Bean  
 public LocalSessionFactoryBean factoryBean(DataSource dataSource) {  
 LocalSessionFactoryBean sessionFactoryBean = new LocalSessionFactoryBean();  
 sessionFactoryBean.setDataSource(dataSource);  
 sessionFactoryBean.setPackagesToScan("prac15.model");  
 Properties properties = new Properties();  
 properties.setProperty("hibernate.dialect", "org.hibernate.dialect.PostgreSQLDialect");  
 sessionFactoryBean.setHibernateProperties(properties);  
 return sessionFactoryBean;  
 }  
 @Bean  
 public PlatformTransactionManager platformTransactionManager(LocalSessionFactoryBean factoryBean){  
 HibernateTransactionManager transactionManager = new HibernateTransactionManager();  
 transactionManager.setSessionFactory(factoryBean.getObject());  
 return transactionManager;  
 }  
  
}

**DepartureController.java**

package prac15.controllers;  
  
import prac15.model.Departure;  
import prac15.service.DepartureService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RequestMapping("/home/departure")  
@Controller  
public class DepartureController {  
  
 @Autowired  
 private DepartureService departureService;  
  
 @GetMapping("/add")  
 @ResponseBody  
 public String add(@RequestParam String type, @RequestParam String departureDate){  
 Departure departure = new Departure();  
 departure.setType(type);  
 departure.setDepartureDate(departureDate);  
 departureService.addDeparture(departure);  
 return "Добавлен: " + type + " " + departureDate;  
 }  
 @GetMapping("/get")  
 @ResponseBody  
 public List<Departure> get(){  
 return departureService.getDepartures();  
 }  
 @GetMapping("/del/{id}")  
 public String remove(@PathVariable int id){  
 return departureService.deleteDeparture(id);  
 }  
}

**PostOfficeController.java**

package prac15.controllers;  
  
import prac15.model.PostOffice;  
import prac15.service.PostOfficeService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RequestMapping("/home/postoffice")  
@RestController  
public class PostOfficeController {  
 @Autowired  
 private PostOfficeService postOfficeService;  
 @GetMapping("/add")  
 @ResponseBody  
 public String add(@RequestParam String name, @RequestParam String cityName){  
 PostOffice postOffice = new PostOffice();  
 postOffice.setName(name);  
 postOffice.setCityName(cityName);  
 postOfficeService.addPostOffice(postOffice);  
 return "Добавлен: " + name + " " + cityName;  
 }  
 @GetMapping("/get")  
 @ResponseBody  
 public List<PostOffice> get(){  
 return postOfficeService.getPostOffices();  
 }  
 @GetMapping("/del/{id}")  
 public String remove(@PathVariable int id){  
 return postOfficeService.deletePostOffice(id);  
 }  
}

**Departure.java**

package prac15.model;  
  
import lombok.Getter;  
import lombok.Setter;  
import javax.persistence.\*;  
  
@Entity  
@Table(name = "departure")  
@Getter  
@Setter  
public class Departure {  
 @Id  
 private int id;  
 @Column(name = "type")  
 String type;  
 @Column(name = "departuredate")  
 String departureDate;  
}

**PostOffice.java**

package prac15.model;  
  
import lombok.Getter;  
import lombok.Setter;  
  
import javax.persistence.\*;  
import javax.persistence.Entity;  
  
  
@Entity  
@Table(name = "postoffice")  
@Getter  
@Setter  
public class PostOffice {  
 @Id  
 private int id;  
 @Column(name = "name")  
 String name;  
 @Column(name = "cityname")  
 String cityName;  
}

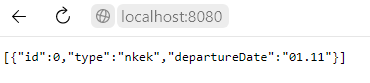
**DepartureService.java**

package prac15.service;  
  
import prac15.model.Departure;  
import lombok.RequiredArgsConstructor;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.springframework.stereotype.Component;  
  
import javax.annotation.PostConstruct;  
import java.util.List;  
  
  
@Component  
@RequiredArgsConstructor  
public class DepartureService {  
 private final SessionFactory sessionFactory;  
 private Session session;  
  
  
 @PostConstruct  
 void init() {  
 session = sessionFactory.openSession();  
 }  
 public List<Departure> getDepartures() {return session.createQuery("select u from Departure u",  
 Departure.class).getResultList();  
 }  
  
 public void addDeparture(Departure departure){  
 Transaction transaction = session.beginTransaction();  
 session.saveOrUpdate(departure);  
 transaction.commit();  
 }  
  
 public String deleteDeparture(int id){  
 Transaction transaction = session.beginTransaction();  
 Departure departure = session.find(Departure.class, id);  
 if( departure == null){  
 return "User is not found";  
 }  
 session.delete(departure);  
 transaction.commit();  
 return "User deleted";  
 }  
}

**PostOfficeService.java**

package prac15.service;  
  
import prac15.model.PostOffice;  
import lombok.RequiredArgsConstructor;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.springframework.stereotype.Component;  
  
import javax.annotation.PostConstruct;  
import java.util.List;  
  
  
@Component  
@RequiredArgsConstructor  
public class PostOfficeService {  
 private final SessionFactory sessionFactory;  
 private Session session;  
  
 @PostConstruct  
 void init() {  
 session = sessionFactory.openSession();  
 }  
 public List<PostOffice> getPostOffices() {return session.createQuery("select u from PostOffice u",  
 PostOffice.class).getResultList();  
 }  
  
 public void addPostOffice(PostOffice postOffice){  
 Transaction transaction = session.beginTransaction();  
 session.saveOrUpdate(postOffice);  
 transaction.commit();  
 }  
  
 public String deletePostOffice(int id){  
 Transaction transaction = session.beginTransaction();  
 PostOffice postOffice = session.find(PostOffice.class, id);  
 if( postOffice == null){  
 return "User is not found";  
 }  
 session.delete(postOffice);  
 transaction.commit();  
 return "User deleted";  
 }  
}

**Результаты:**

****

**Выводы:**

В результате проделанной работы научились работать с базой данных PostgreSQL.

# Практическая работа №16

**Задание:** Создать связь Один-ко-многим между сущностями из предыдущего задания и проверить работу lazy loading.

**Цель работы:** Изучение видов связей между сущностями в Hibernate. Использование транзакций.

**Выполнение работы:**

**DepartureController.java**

package prac16.controllers;  
  
import prac16.model.Departure;  
import prac16.model.PostOffice;  
import prac16.service.DepartureService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RequestMapping("/home/departure")  
@Controller  
public class DepartureController {  
  
 @Autowired  
 private DepartureService departureService;  
  
 @GetMapping("/add")  
 @ResponseBody  
 public String add(@RequestParam int Postoffice\_id, @RequestParam String type, @RequestParam String departureDate){  
 Departure departure = new Departure();  
 departure.setType(type);  
 departure.setPostoffice\_id(Postoffice\_id);  
 departure.setDepartureDate(departureDate);  
 departureService.addDeparture(departure);  
 return "Добавлен: " + type + " " + departureDate;  
 }  
 @GetMapping("/get")  
 @ResponseBody  
 public List<Departure> get(){  
 return departureService.getDepartures();  
 }  
 @GetMapping("/del/{id}")  
 public String remove(@PathVariable int id){  
 return departureService.deleteDeparture(id);  
 }  
  
 @GetMapping(value = "/{postofficeid}/postoffice")  
 @ResponseBody  
 public PostOffice getPostUser(@PathVariable("postofficeid") int postofficeid){  
 return departureService.getPostOfficeByDeparture(postofficeid);  
 }  
}

**PostOfficeController.java**

package prac16.controllers;  
  
import prac16.model.PostOffice;  
import prac16.service.PostOfficeService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RequestMapping("/home/postoffice")  
@RestController  
public class PostOfficeController {  
 @Autowired  
 private PostOfficeService postOfficeService;  
 @GetMapping("/add")  
 @ResponseBody  
 public String add(@RequestParam String name, @RequestParam String cityName){  
 PostOffice postOffice = new PostOffice();  
 postOffice.setName(name);  
 postOffice.setCityName(cityName);  
 postOfficeService.addPostOffice(postOffice);  
 return "Добавлен: " + name + " " + cityName;  
 }  
 @GetMapping("/get")  
 @ResponseBody  
 public List<PostOffice> get(){  
 return postOfficeService.getPostOffices();  
 }  
 @GetMapping("/del/{id}")  
 public String remove(@PathVariable int id){  
 return postOfficeService.deletePostOffice(id);  
 }  
}

**Departure.java**

package prac16.model;  
  
import com.fasterxml.jackson.annotation.JsonIgnore;  
import lombok.Getter;  
import lombok.Setter;  
  
import javax.persistence.\*;  
  
@Entity  
@Table(name = "departures")  
@Getter  
@Setter  
public class Departure {  
 @Id  
 @SequenceGenerator(name = "departures\_seq", sequenceName = "departures\_sequence", allocationSize = 1)  
 @GeneratedValue(generator = "departures\_seq", strategy = GenerationType.SEQUENCE)  
 private int id;  
 @Column(name = "type")  
 String type;  
 @Column(name = "departuredate")  
 String departureDate;  
 @Column(name = "postoffice\_id")  
 private int postoffice\_id;  
 @ManyToOne  
 @JoinColumn(name = "postoffice\_id", insertable = false, updatable = false)  
 @JsonIgnore  
 private PostOffice postoffice;  
}

**PostOffice.java**

package prac16.model;  
  
import lombok.Getter;  
import lombok.Setter;  
  
import javax.persistence.\*;  
import java.util.List;  
  
  
@Entity  
@Table(name = "postoffices")  
@Getter  
@Setter  
public class PostOffice {  
 @Id  
 @SequenceGenerator(name = "postoffice\_seq", sequenceName = "postoffice\_sequence", allocationSize = 1)  
 @GeneratedValue(generator = "postoffice\_seq", strategy = GenerationType.SEQUENCE)  
 private int id;  
 @Column(name = "name")  
 String name;  
 @Column(name = "cityname")  
 String cityName;  
 @OneToMany(mappedBy = "postoffice", fetch = FetchType.LAZY)  
 private List<Departure> departures;  
}

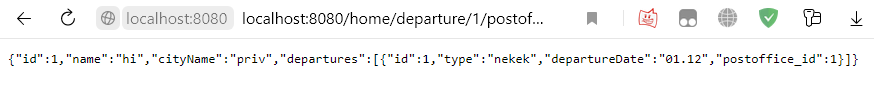
**DepartureService.java**

package prac16.service;  
  
import lombok.RequiredArgsConstructor;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.springframework.stereotype.Component;  
import prac16.model.Departure;  
import prac16.model.PostOffice;  
  
import javax.annotation.PostConstruct;  
import java.util.List;  
  
  
@Component  
@RequiredArgsConstructor  
public class DepartureService {  
 private final SessionFactory sessionFactory;  
 private Session session;  
  
  
 @PostConstruct  
 void init() {  
 session = sessionFactory.openSession();  
 }  
 public List<Departure> getDepartures() {return session.createQuery("select u from Departure u",  
 Departure.class).getResultList();  
 }  
  
 public void addDeparture(Departure departure){  
 Transaction transaction = session.beginTransaction();  
 session.saveOrUpdate(departure);  
 transaction.commit();  
 }  
  
 public String deleteDeparture(int id){  
 Transaction transaction = session.beginTransaction();  
 Departure departure = session.find(Departure.class, id);  
 if( departure == null){  
 return "Departure is not found";  
 }  
 session.delete(departure);  
 transaction.commit();  
 return "Departure deleted";  
 }  
 public PostOffice getPostOfficeByDeparture(int postOfficeId) {  
 return session.createQuery("FROM Departure where id = :id", Departure.class)  
 .setParameter("id", postOfficeId).getSingleResult().getPostoffice();  
 }  
}

**PostOfficeService.java**

package prac16.service;  
  
import prac16.model.PostOffice;  
import lombok.RequiredArgsConstructor;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.springframework.stereotype.Component;  
  
import javax.annotation.PostConstruct;  
import java.util.List;  
  
  
@Component  
@RequiredArgsConstructor  
public class PostOfficeService {  
 private final SessionFactory sessionFactory;  
 private Session session;  
  
 @PostConstruct  
 void init() {  
 session = sessionFactory.openSession();  
 }  
 public List<PostOffice> getPostOffices() {return session.createQuery("select u from PostOffice u",  
 PostOffice.class).getResultList();  
 }  
  
 public void addPostOffice(PostOffice postOffice){  
 Transaction transaction = session.beginTransaction();  
 session.saveOrUpdate(postOffice);  
 transaction.commit();  
 }  
  
 public String deletePostOffice(int id){  
 Transaction transaction = session.beginTransaction();  
 PostOffice postOffice = session.find(PostOffice.class, id);  
 if( postOffice == null){  
 return "PostOffice is not found";  
 }  
 session.delete(postOffice);  
 transaction.commit();  
 return "PostOffice deleted";  
 }  
}

**Результаты:**

****

**Выводы:**

В результате данной работы реализована связь Один-ко-многим между сущностями в Java, используя базу данных PostgreSQL.

# Практическая работа №17

**Задание:** Добавить возможность фильтрации по всем полям всех классов с использованием Criteria API в Hibernate для программы из предыдущего задания. Добавить эндпоинты для каждой фильтрации.

**Цель работы:** Знакомство с Criteria API в Hibernate.

**Выполнение работы:**

**DepartureController.java**

package prac17.controllers;  
  
import prac17.model.Departure;  
import prac17.model.PostOffice;  
import prac17.service.DepartureService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RequestMapping("/home/departure")  
@Controller  
public class DepartureController {  
  
 @Autowired  
 private DepartureService departureService;  
  
 @GetMapping("/add")  
 @ResponseBody  
 public String add(@RequestParam int Postoffice\_id, @RequestParam String type, @RequestParam String departureDate){  
 Departure departure = new Departure();  
 departure.setType(type);  
 departure.setPostoffice\_id(Postoffice\_id);  
 departure.setDepartureDate(departureDate);  
 departureService.addDeparture(departure);  
 return "Добавлен: " + type + " " + departureDate;  
 }  
 @GetMapping("/get")  
 @ResponseBody  
 public List<Departure> get(){  
 return departureService.getDepartures();  
 }  
 @GetMapping("/del/{id}")  
 public String remove(@PathVariable int id){  
 return departureService.deleteDeparture(id);  
 }  
  
 @GetMapping(value = "/{postofficeid}/postoffice")  
 @ResponseBody  
 public PostOffice getPostUser(@PathVariable("postofficeid") int postofficeid){  
 return departureService.getPostOfficeByDeparture(postofficeid);  
 }  
 @GetMapping("/get/sorted/type")  
 @ResponseBody  
 public List<Departure> getSortedDeparturesByType(){  
 return departureService.getSortedDeparturesByType();  
 }  
 @GetMapping("/get/sorted/departuredate")  
 @ResponseBody  
 public List<Departure> getSortedDeparturesByDepartureDate(){  
 return departureService.getSortedDeparturesByDepartureDate();  
 }  
}

**PostOfficeController.java**

package prac17.controllers;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
import prac17.model.PostOffice;  
import prac17.service.PostOfficeService;  
  
import java.util.List;  
  
@RequestMapping("/home/postoffice")  
@RestController  
public class PostOfficeController {  
 @Autowired  
 private PostOfficeService postOfficeService;  
 @GetMapping("/add")  
 @ResponseBody  
 public String add(@RequestParam String name, @RequestParam String cityName){  
 PostOffice postOffice = new PostOffice();  
 postOffice.setName(name);  
 postOffice.setCityName(cityName);  
 postOfficeService.addPostOffice(postOffice);  
 return "Добавлен: " + name + " " + cityName;  
 }  
 @GetMapping("/get")  
 @ResponseBody  
 public List<PostOffice> get(){  
 return postOfficeService.getPostOffices();  
 }  
 @GetMapping("/del/{id}")  
 public String remove(@PathVariable int id){  
 return postOfficeService.deletePostOffice(id);  
 }  
 @GetMapping("/get/sorted/cityname")  
 @ResponseBody  
 public List<PostOffice> getSortedPostOfficesByCityName(){  
 return postOfficeService.getSortedPostOfficesByCityName();  
 }  
 @GetMapping("/get/sorted/name")  
 @ResponseBody  
 public List<PostOffice> getSortedPostOfficesByName(){  
 return postOfficeService.getSortedPostOfficesByName();  
 }  
}

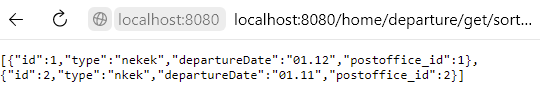
**DepartureService.java**

package prac17.service;  
  
import lombok.RequiredArgsConstructor;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.hibernate.query.Query;  
import org.springframework.stereotype.Component;  
import prac17.model.Departure;  
import prac17.model.PostOffice;  
  
import javax.annotation.PostConstruct;  
import javax.persistence.criteria.CriteriaBuilder;  
import javax.persistence.criteria.CriteriaQuery;  
import javax.persistence.criteria.Root;  
import java.util.List;  
  
  
@Component  
@RequiredArgsConstructor  
public class DepartureService {  
 private final SessionFactory sessionFactory;  
 private Session session;  
  
  
 @PostConstruct  
 void init() {  
 session = sessionFactory.openSession();  
 }  
 public List<Departure> getDepartures() {return session.createQuery("select u from Departure u",  
 Departure.class).getResultList();  
 }  
  
 public void addDeparture(Departure departure){  
 Transaction transaction = session.beginTransaction();  
 session.saveOrUpdate(departure);  
 transaction.commit();  
 }  
  
 public String deleteDeparture(int id){  
 Transaction transaction = session.beginTransaction();  
 Departure departure = session.find(Departure.class, id);  
 if( departure == null){  
 return "Departure is not found";  
 }  
 session.delete(departure);  
 transaction.commit();  
 return "Departure deleted";  
 }  
 public PostOffice getPostOfficeByDeparture(int postOfficeId) {  
 return session.createQuery("FROM Departure where id = :id", Departure.class)  
 .setParameter("id", postOfficeId).getSingleResult().getPostoffice();  
 }  
  
 public List<Departure> getSortedDeparturesByType(){  
 CriteriaBuilder builder = session.getCriteriaBuilder();  
 CriteriaQuery<Departure> departureCriteriaQuery = builder.createQuery(Departure.class);  
 Root<Departure> root = departureCriteriaQuery.from(Departure.class);  
 departureCriteriaQuery.select(root).orderBy(builder.asc(root.get("type")));  
 Query<Departure> query = session.createQuery(departureCriteriaQuery);  
 return query.getResultList();  
 }  
  
 public List<Departure> getSortedDeparturesByDepartureDate(){  
 CriteriaBuilder builder = session.getCriteriaBuilder();  
 CriteriaQuery<Departure> departureCriteriaQuery = builder.createQuery(Departure.class);  
 Root<Departure> root = departureCriteriaQuery.from(Departure.class);  
 departureCriteriaQuery.select(root).orderBy(builder.asc(root.get("departuredate")));  
 Query<Departure> query = session.createQuery(departureCriteriaQuery);  
 return query.getResultList();  
 }  
  
}

**PostOfficeService.java**

package prac17.service;  
  
import org.hibernate.query.Query;  
import prac17.model.PostOffice;  
import lombok.RequiredArgsConstructor;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.springframework.stereotype.Component;  
  
import javax.annotation.PostConstruct;  
import javax.persistence.criteria.CriteriaBuilder;  
import javax.persistence.criteria.CriteriaQuery;  
import javax.persistence.criteria.Root;  
import java.util.List;  
  
  
@Component  
@RequiredArgsConstructor  
public class PostOfficeService {  
 private final SessionFactory sessionFactory;  
 private Session session;  
  
 @PostConstruct  
 void init() {  
 session = sessionFactory.openSession();  
 }  
 public List<PostOffice> getPostOffices() {return session.createQuery("select u from PostOffice u",  
 PostOffice.class).getResultList();  
 }  
  
 public void addPostOffice(PostOffice postOffice){  
 Transaction transaction = session.beginTransaction();  
 session.saveOrUpdate(postOffice);  
 transaction.commit();  
 }  
  
 public String deletePostOffice(int id){  
 Transaction transaction = session.beginTransaction();  
 PostOffice postOffice = session.find(PostOffice.class, id);  
 if( postOffice == null){  
 return "PostOffice is not found";  
 }  
 session.delete(postOffice);  
 transaction.commit();  
 return "PostOffice deleted";  
 }  
  
 public List<PostOffice> getSortedPostOfficesByName(){  
 CriteriaBuilder builder = session.getCriteriaBuilder();  
 CriteriaQuery<PostOffice> postOfficeCriteriaQuery = builder.createQuery(PostOffice.class);  
 Root<PostOffice> root = postOfficeCriteriaQuery.from(PostOffice.class);  
 postOfficeCriteriaQuery.select(root).orderBy(builder.asc(root.get("name")));  
 Query<PostOffice> query = session.createQuery(postOfficeCriteriaQuery);  
 return query.getResultList();  
 }  
  
 public List<PostOffice> getSortedPostOfficesByCityName(){  
 CriteriaBuilder builder = session.getCriteriaBuilder();  
 CriteriaQuery<PostOffice> postOfficeCriteriaQuery = builder.createQuery(PostOffice.class);  
 Root<PostOffice> root = postOfficeCriteriaQuery.from(PostOffice.class);  
 postOfficeCriteriaQuery.select(root).orderBy(builder.asc(root.get("cityname")));  
 Query<PostOffice> query = session.createQuery(postOfficeCriteriaQuery);  
 return query.getResultList();  
 }  
}

**Результаты:**

****

**Выводы:** Проделав данную работу, мы освоили Criteria API в Hibernate.

# Практическая работа №18

**Задание:** Переписать код предыдущего задания с использованием сервисов и отделения логики контроллера от логики сервиса и репозитория. В программе всё взаимодействие с базой данных должно быть реализовано через репозитории Spring Data Jpa.

**Цель работы:** Знакомство с репозиториями и сервисами, реализация в проекте. Взаимодействие с Spring Data JPA.

**Выполнение работы:**

**Config.java**

package prac18;  
  
import org.springframework.context.annotation.Configuration;  
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;  
  
@Configuration  
@EnableJpaRepositories  
public class Config {  
}

**DepartureRepository.java**

package prac18.repositories;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import prac18.model.Departure;  
  
public interface DepartureRepository extends JpaRepository<Departure, Integer> { }

**PostOfficeRepository.java**

package prac18.repositories;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import prac18.model.PostOffice;  
  
public interface PostOfficeRepository extends JpaRepository<PostOffice, Integer> { }

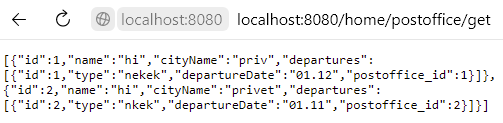
**DepartureService.java**

package prac18.service;  
  
import lombok.RequiredArgsConstructor;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Service;  
import prac18.model.Departure;  
import prac18.model.PostOffice;  
import prac18.repositories.DepartureRepository;  
  
import java.util.List;  
  
  
@Service  
@RequiredArgsConstructor  
public class DepartureService {  
 private final DepartureRepository departureRepository;  
  
 public List<Departure> getDepartures() {return departureRepository.findAll();}  
  
 public void addDeparture(Departure departure){  
 departureRepository.save(departure);}  
  
 public String deleteDeparture(int id){  
 if( departureRepository.existsById(id)){  
 return "Departure deleted";  
 }  
 return "Departure is not found";  
 }  
 public PostOffice getPostOfficeByDeparture(int postOfficeId) {  
 return departureRepository.findById(postOfficeId).orElseThrow(() ->  
 new IllegalStateException("Post with this id notfound")).getPostoffice();  
 }  
  
 public List<Departure> getSortedDeparturesByType(){return departureRepository.findAll(Sort.by("type")); }  
  
 public List<Departure> getSortedDeparturesByDepartureDate(){  
 return departureRepository.findAll(Sort.by("departuredate"));  
 }  
  
}

**PostOfficeService.java**

package prac18.service;  
  
import lombok.RequiredArgsConstructor;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Service;  
import prac18.model.PostOffice;  
import prac18.repositories.PostOfficeRepository;  
  
import java.util.List;  
  
  
@Service  
@RequiredArgsConstructor  
public class PostOfficeService {  
 private final PostOfficeRepository postOfficeRepository;  
  
 public List<PostOffice> getPostOffices() {return postOfficeRepository.findAll(); }  
  
 public void addPostOffice(PostOffice postOffice){ postOfficeRepository.save(postOffice); }  
  
 public String deletePostOffice(int id){  
 if( postOfficeRepository.existsById(id)){  
 return "PostOffice deleted";  
 }  
 return "PostOffice is not found";  
 }  
  
 public List<PostOffice> getSortedPostOfficesByName(){ return postOfficeRepository.findAll(Sort.by("name")); }  
  
 public List<PostOffice> getSortedPostOfficesByCityName(){  
 return postOfficeRepository.findAll(Sort.by("cityname"));  
 }  
}

**Результаты:**

****

**Выводы:** В ходе данной практической работы мы познакомились с Spring Data Jpa и реализовали работу логики контроллера отдельно от логики сервиса и репозитория, а также переписали взаиможействие с базой данных через Spring Data Jpa.

# Практическая работа №19

**Задание:** Создать файл logback.xml, добавить логирование во все методы классов-сервисов.

**Цель работы:** Знакомство с логированием с использованием Logback в Spring.

**Выполнение работы:**

**DepartureService.java**

package prac19.service;  
  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Service;  
import prac19.model.Departure;  
import prac19.model.PostOffice;  
import prac19.repositories.DepartureRepository;  
  
import java.util.List;  
  
  
@Service  
@RequiredArgsConstructor  
@Slf4j  
public class DepartureService {  
 private final DepartureRepository departureRepository;  
  
 public List<Departure> getDepartures() {  
 log.info("Get all departures");  
 return departureRepository.findAll();  
 }  
  
 public void addDeparture(Departure departure){  
 log.info("Add departure");  
 departureRepository.save(departure);  
 }  
  
 public String deleteDeparture(int id){  
 if( departureRepository.existsById(id)){  
 log.debug("Delete departure");  
 return "Departure deleted";  
 }  
 log.debug("Delete departure. Can not find departure");  
 return "Departure is not found";  
 }  
 public PostOffice getPostOfficeByDeparture(int postOfficeId) {  
 log.info("Get post office by departure");  
 return departureRepository.findById(postOfficeId).orElseThrow(() ->  
 new IllegalStateException("Post with this id notfound")).getPostoffice();  
 }  
  
 public List<Departure> getSortedDeparturesByType(){  
 log.info("Get sorted departures by type");  
 return departureRepository.findAll(Sort.by("type"));  
 }  
  
 public List<Departure> getSortedDeparturesByDepartureDate(){  
 log.info("Get sorted departures by departure date");  
 return departureRepository.findAll(Sort.by("departuredate"));  
 }  
  
}

**PostOfficeService.java**

package prac19.service;  
  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Service;  
import prac19.model.PostOffice;  
import prac19.repositories.PostOfficeRepository;  
  
import java.util.List;  
  
  
@Service  
@RequiredArgsConstructor  
@Slf4j  
public class PostOfficeService {  
 private final PostOfficeRepository postOfficeRepository;  
  
 public List<PostOffice> getPostOffices() {  
 log.info("Get all post offices");  
 return postOfficeRepository.findAll();  
 }  
  
 public void addPostOffice(PostOffice postOffice){  
 log.info("Add post office");  
 postOfficeRepository.save(postOffice);  
 }  
  
 public String deletePostOffice(int id){  
 if( postOfficeRepository.existsById(id)){  
 log.debug("Delete post office");  
 return "PostOffice deleted";  
 }  
 log.debug("Delete post office. Can not find post office");  
 return "PostOffice is not found";  
 }  
  
 public List<PostOffice> getSortedPostOfficesByName(){  
 log.info("Get sorted post office by name");  
 return postOfficeRepository.findAll(Sort.by("name"));  
 }  
  
 public List<PostOffice> getSortedPostOfficesByCityName(){  
  
 log.info("Get sorted post office by cityname");  
 return postOfficeRepository.findAll(Sort.by("cityname"));  
 }  
}

**Результаты:**

****

**Выводы:**

В результате данной работы реализовано логирование в текущем проекте.

# Практическая работа №20

**Задание:** Для приложения из предыдущего задания добавить логирование времени выполнения каждого метода сервиса с использованием Spring AOP.

**Цель работы:** Использование Spring AOP. Pointcut, JoinPoint. Advice.

**Выполнение работы:**

**Aspect.java**

package prac20;  
  
import lombok.extern.slf4j.Slf4j;  
import org.aspectj.lang.ProceedingJoinPoint;  
import org.aspectj.lang.annotation.Around;  
import org.aspectj.lang.annotation.Pointcut;  
import org.springframework.stereotype.Component;  
  
import java.util.Arrays;  
  
@Slf4j  
@Component  
@org.aspectj.lang.annotation.Aspect  
public class Aspect {  
 @Around("allServiceMethods()")  
 public Object logParameters(ProceedingJoinPoint joinPoint) {  
 long start = System.currentTimeMillis();  
 Object result = null;  
 try {  
 result = joinPoint.proceed();  
 } catch (Throwable throwable) {  
 throwable.printStackTrace();  
 }  
 long end = System.currentTimeMillis();  
 long time = end - start;  
 log.info("Method: " + joinPoint.getSignature().getName() + " from " + joinPoint.getTarget().getClass() +  
 ", Time: " + time + " ms, Parameters: " + Arrays.toString(joinPoint.getArgs()));  
 return result;  
 }  
 @Pointcut("within(prac20.service.\*)")  
 public void allServiceMethods() {}  
}

**DepartureService.java**

package prac20.service;  
  
import lombok.RequiredArgsConstructor;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Service;  
import prac20.model.Departure;  
import prac20.model.PostOffice;  
import prac20.repositories.DepartureRepository;  
  
import java.util.List;  
  
  
@Service  
@RequiredArgsConstructor  
public class DepartureService {  
 private final DepartureRepository departureRepository;  
  
 public List<Departure> getDepartures() { return departureRepository.findAll(); }  
  
 public void addDeparture(Departure departure){ departureRepository.save(departure); }  
  
 public String deleteDeparture(int id){  
 if( departureRepository.existsById(id)){  
 return "Departure deleted";  
 }  
 return "Departure is not found";  
 }  
 public PostOffice getPostOfficeByDeparture(int postOfficeId) {  
 return departureRepository.findById(postOfficeId).orElseThrow(() ->  
 new IllegalStateException("Post with this id notfound")).getPostoffice();  
 }  
  
 public List<Departure> getSortedDeparturesByType(){ return departureRepository.findAll(Sort.by("type")); }  
  
 public List<Departure> getSortedDeparturesByDepartureDate(){  
 return departureRepository.findAll(Sort.by("departuredate"));  
 }  
  
}

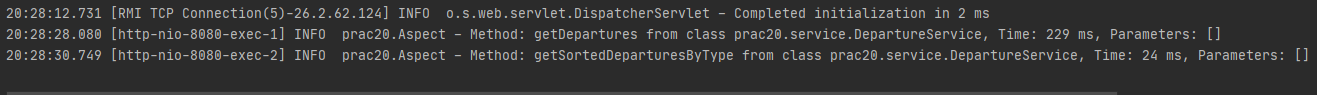
**PostOfficeService.java**

package prac20.service;  
  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Service;  
import prac20.model.PostOffice;  
import prac20.repositories.PostOfficeRepository;  
  
import java.util.List;  
  
  
@Service  
@RequiredArgsConstructor  
@Slf4j  
public class PostOfficeService {  
 private final PostOfficeRepository postOfficeRepository;  
  
 public List<PostOffice> getPostOffices() { return postOfficeRepository.findAll(); }  
  
 public void addPostOffice(PostOffice postOffice){ postOfficeRepository.save(postOffice); }  
  
 public String deletePostOffice(int id){  
 if( postOfficeRepository.existsById(id)){  
 return "PostOffice deleted";  
 }  
 return "PostOffice is not found";  
 }  
  
 public List<PostOffice> getSortedPostOfficesByName(){ return postOfficeRepository.findAll(Sort.by("name")); }  
  
 public List<PostOffice> getSortedPostOfficesByCityName(){  
 return postOfficeRepository.findAll(Sort.by("cityname"));  
 }  
}

**logback.xml**

<configuration>  
 <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">  
 <encoder>  
 <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} – %msg%n</pattern>  
 </encoder>  
 </appender>  
 <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">  
 <file>application.log</file>  
 <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
 <fileNamePattern>application.%d{yyyy-MM-dd}.gz</fileNamePattern>  
 <maxHistory>30</maxHistory>  
 <totalSizeCap>3GB</totalSizeCap>  
 </rollingPolicy>  
 <encoder>  
 <pattern>%-4relative [%thread] %-5level %logger{35} – %msg%n</pattern>  
 </encoder>  
 </appender>  
 <root level="info">  
 <appender-ref ref="STDOUT"/>  
 <appender-ref ref="FILE"/>  
 </root>  
</configuration>

**Результаты:**

****

**Выводы:**

Проделав данную работу, мы освоили использование Spring AOP, Pointcut, JoinPoint, Advice.

# Практическая работа №21

**Задание:** Для приложения из предыдущего задания пометить все классы сервисов, в которых происходит взаимодействие с базой данных, как Transactional. Добавить отправку информации о сохранении каждого объекта по электронной почте, создав отдельный класс EmailService с асинхронными методами отправки сообщений. Для асинхронности методов используйте аннотацию Async.

**Цель работы:** Проксирование. Аннотация Transactional. Аннотация Async.

**Выполнение работы:**

**EmailConfig.java**

package prac21.Configs;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.mail.javamail.JavaMailSender;  
import org.springframework.mail.javamail.JavaMailSenderImpl;  
  
import java.util.Properties;  
  
@Configuration  
public class EmailConfig {  
 @Value("${spring.mail.username}")  
 private String username;  
 @Value("${spring.mail.password}")  
 private String password;  
 @Bean  
 public JavaMailSender getMailSender(){  
 JavaMailSenderImpl mailSender = new JavaMailSenderImpl();  
 mailSender.setHost("smtp.gmail.com");  
 mailSender.setPort(587);  
  
 mailSender.setUsername(username);  
 mailSender.setPassword(password);  
  
 Properties props = mailSender.getJavaMailProperties();  
 props.put("mail.transport.protocol", "smtp");  
 props.put("mail.smtp.auth", "true");  
 props.put("mail.smtp.starttls.enable", "true");  
 props.put("mail.debug", "true");  
  
 return mailSender;  
 }  
}

**DepartureService.java**

package prac21.service;  
  
import lombok.RequiredArgsConstructor;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import prac21.model.Departure;  
import prac21.model.PostOffice;  
import prac21.repositories.DepartureRepository;  
  
import java.util.List;  
  
  
@Service  
@RequiredArgsConstructor  
public class DepartureService {  
 private final DepartureRepository departureRepository;  
  
 @Autowired  
 private EmailService emailService;  
  
 @Transactional(readOnly = true)  
 public List<Departure> getDepartures() { return departureRepository.findAll(); }  
  
 @Transactional  
 public void addDeparture(Departure departure){  
 emailService.sendMail("Add new departure","Add new departure: " + departure.getType());  
 departureRepository.save(departure);  
 }  
  
 @Transactional  
 public String deleteDeparture(int id){  
 if( departureRepository.existsById(id)){  
 return "Departure deleted";  
 }  
 return "Departure is not found";  
 }  
  
 @Transactional(readOnly = true)  
 public PostOffice getPostOfficeByDeparture(int postOfficeId) {  
 return departureRepository.findById(postOfficeId).orElseThrow(() ->  
 new IllegalStateException("Post with this id notfound")).getPostoffice();  
 }  
  
 @Transactional(readOnly = true)  
 public List<Departure> getSortedDeparturesByType(){ return departureRepository.findAll(Sort.by("type")); }  
  
 @Transactional(readOnly = true)  
 public List<Departure> getSortedDeparturesByDepartureDate(){  
 return departureRepository.findAll(Sort.by("departuredate"));  
 }  
  
}

**EmailService.java**

package prac21.service;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.mail.SimpleMailMessage;  
import org.springframework.mail.javamail.JavaMailSender;  
import org.springframework.scheduling.annotation.Async;  
import org.springframework.stereotype.Service;  
  
@Service  
public class EmailService {  
 @Autowired  
 private JavaMailSender mailSender;  
 @Value("${spring.mail.username}")  
 private String setFrom;  
 @Value("${ADDRESS.TO}")  
 private String setTo;  
 @Async  
 public void sendMail(String subject, String text){  
 SimpleMailMessage message = new SimpleMailMessage();  
 message.setFrom(setFrom);  
 message.setTo(setTo);  
 message.setSubject(subject);  
 message.setText(text);  
 mailSender.send(message);  
 }  
}

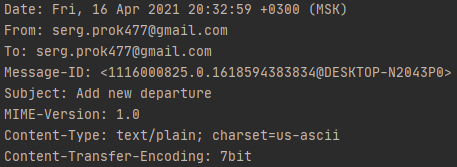
**PostOfficeService.java**

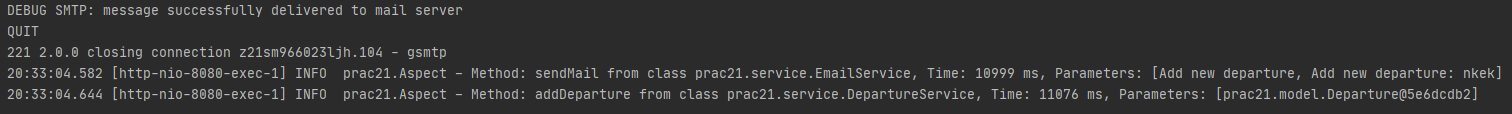
package prac21.service;  
  
import lombok.RequiredArgsConstructor;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import prac21.model.PostOffice;  
import prac21.repositories.PostOfficeRepository;  
  
import java.util.List;  
  
  
@Service  
@RequiredArgsConstructor  
public class PostOfficeService {  
 private final PostOfficeRepository postOfficeRepository;  
  
 @Autowired  
 private EmailService emailService;  
  
 @Transactional(readOnly = true)  
 public List<PostOffice> getPostOffices() { return postOfficeRepository.findAll(); }  
  
 @Transactional  
 public void addPostOffice(PostOffice postOffice){  
 emailService.sendMail("Add new post office","Add new post office: " + postOffice.getName());  
 postOfficeRepository.save(postOffice);  
 }  
  
 @Transactional  
 public String deletePostOffice(int id){  
 if( postOfficeRepository.existsById(id)){  
 return "PostOffice deleted";  
 }  
 return "PostOffice is not found";  
 }  
  
 @Transactional(readOnly = true)  
 public List<PostOffice> getSortedPostOfficesByName(){ return postOfficeRepository.findAll(Sort.by("name")); }  
  
 @Transactional(readOnly = true)  
 public List<PostOffice> getSortedPostOfficesByCityName(){  
 return postOfficeRepository.findAll(Sort.by("cityname"));  
 }  
}

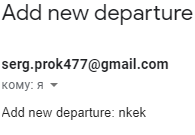
**application.properties**

spring.mail.host=smtp.gmail.com  
spring.mail.port=587  
spring.mail.username=${ADDRESS.FROM}  
spring.mail.password=${ADDRESS.PASSWORD}  
spring.mail.properties.mail.smtp.auth=true  
spring.mail.properties.mail.smtp.starttls.enable=true

**Результаты:**

****

****

****

**Выводы:**

В ходе данной практической работы освоена работа с асинхронными вызовами, проксирование, планирование заданий в Spring.

# Практическая работа №22

**Задание:** Для приложения из предыдущего задания создать класс-сервис с методом, который будет вызываться каждые 30 минут и очищать определённую директорию, а затем создавать по файлу для каждой из сущностей и загружать туда все данные из базы данных. Также добавить возможность вызывать данный метод с использованием Java Management Extensions (JMX).

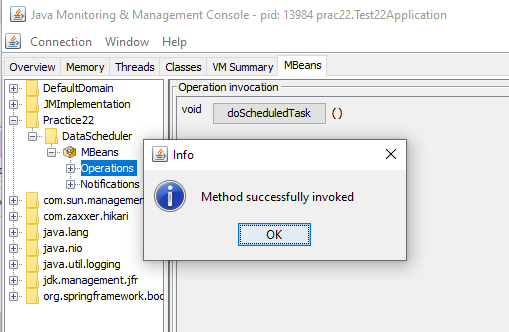
**Цель работы:** Планирование заданий. Scheduler в Spring.

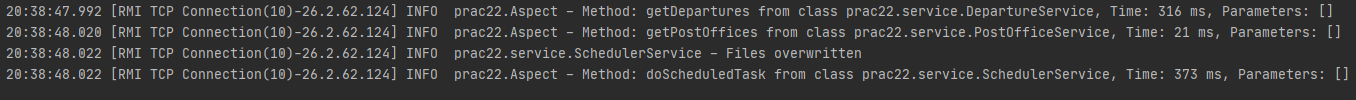
**Выполнение работы:**

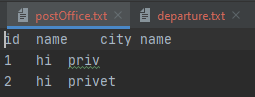
**SchedulerService.java**

package prac22.service;  
  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.jmx.export.annotation.ManagedOperation;  
import org.springframework.jmx.export.annotation.ManagedResource;  
import org.springframework.scheduling.annotation.Scheduled;  
import org.springframework.stereotype.Service;  
import org.springframework.util.ResourceUtils;  
import prac22.model.Departure;  
import prac22.model.PostOffice;  
import java.io.BufferedWriter;  
import java.io.File;  
import java.io.FileWriter;  
import java.io.IOException;  
@Service  
@Slf4j  
@ManagedResource(  
 objectName = "Practice22:category=MBeans,name=DataScheduler"  
)  
public class SchedulerService {  
  
 @Autowired  
 private PostOfficeService postOfficeService;  
 @Autowired  
 private DepartureService departureService;  
 @Scheduled(cron = "0 0/30 \* \* \* \*")  
 @ManagedOperation(description = "Overwrite files in out")  
 public void doScheduledTask() throws IOException {  
 System.out.println("Scheduled task");  
 File file\_dir = ResourceUtils.getFile("Java/sem\_4/prac\_22/src/main/resources/out");  
 try {  
 for (File file : file\_dir.listFiles())  
 if (file.isFile()) file.delete();  
 } catch (NullPointerException e) {  
 e.printStackTrace();  
 }  
 BufferedWriter departureFile = new BufferedWriter(new FileWriter(file\_dir.getAbsolutePath()  
 + "/departure.txt"));  
 BufferedWriter postOfficeFile = new BufferedWriter(new FileWriter(file\_dir.getAbsolutePath()  
 + "/postOffice.txt"));  
 departureFile.write("id\tpostoffice\_id\ttype\tdeparture date\n");  
 for (Departure departure : departureService.getDepartures())  
 departureFile.write(String.format("%d\t%d\t%s\t%s\n", departure.getId(), departure.getPostoffice\_id(),  
 departure.getType(), departure.getDepartureDate()));  
 departureFile.close();  
  
 postOfficeFile.write("id\tname\tcity name\n");  
 for (PostOffice postOffice : postOfficeService.getPostOffices())  
 postOfficeFile.write(String.format("%d\t%s\t%s\n", postOffice.getId(), postOffice.getName(),  
 postOffice.getCityName()));  
 postOfficeFile.close();  
 log.info("Files overwritten");  
 }  
}

**Результаты:**

****

****

****

**Выводы:**

В результате данной работы реализован класс-сервис "Scheduler" для обработки запланированных заданий в Spring.

# Практическая работа №23

**Задание:** В приложении из предыдущего задания добавить возможность регистрации и авторизации пользователей, хранение cookie сессий в базе данных PostgreSQL, хеширование паролей алгоритмом Bcrypt, защиту всех запросов, кроме запросов на авторизацию и регистрацию, от неавторизированных пользователей.

**Цель работы:** Использование Spring Security для аутентификации и авторизации пользователей.

**Выполнение работы:**

**SecurityConfig.java**

package prac23.Configs;  
  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;  
import prac23.service.UserService;  
  
@Configuration  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Autowired  
 private UserService userService;  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 .authorizeRequests()  
 .antMatchers(  
 "/registration\*\*",  
 "/js/\*\*",  
 "/css/\*\*",  
 "/img/\*\*",  
 "/webjars/\*\*").permitAll()  
 .anyRequest().authenticated()  
 .and()  
 .formLogin()  
 .loginPage("/login")  
 .permitAll()  
 .and()  
 .logout()  
 .invalidateHttpSession(true)  
 .clearAuthentication(true)  
 .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))  
 .logoutSuccessUrl("/login?logout")  
 .permitAll();  
 }  
  
 @Bean  
 public BCryptPasswordEncoder bCryptPasswordEncoder(){  
 return new BCryptPasswordEncoder();  
 }  
  
  
 @Autowired  
 protected void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {  
 auth.userDetailsService(userService).passwordEncoder(bCryptPasswordEncoder());  
 }  
}

**FieldMatch.java**

package prac23.constraint;  
  
import javax.validation.Payload;  
import javax.validation.Constraint;  
import java.lang.annotation.Documented;  
import java.lang.annotation.Retention;  
import java.lang.annotation.Target;  
import static java.lang.annotation.ElementType.ANNOTATION\_TYPE;  
import static java.lang.annotation.ElementType.TYPE;  
import static java.lang.annotation.RetentionPolicy.RUNTIME;  
  
@Target({TYPE, ANNOTATION\_TYPE})  
@Retention(RUNTIME)  
@Constraint(validatedBy = FieldMatchValidator.class)  
@Documented  
public @interface FieldMatch  
{  
 String message() default "{constraints.field-match}";  
 Class<?>[] groups() default {};  
 Class<? extends Payload>[] payload() default {};  
 String first();  
 String second();  
  
 @Target({TYPE, ANNOTATION\_TYPE})  
 @Retention(RUNTIME)  
 @Documented  
 @interface List  
 {  
 FieldMatch[] value();  
 }  
}

**FieldMatchValidator.java**

package prac23.constraint;  
  
import org.apache.commons.beanutils.BeanUtils;  
import javax.validation.ConstraintValidator;  
import javax.validation.ConstraintValidatorContext;  
  
public class FieldMatchValidator implements ConstraintValidator<FieldMatch, Object> {  
  
 private String firstFieldName;  
 private String secondFieldName;  
  
 @Override  
 public void initialize(final FieldMatch constraintAnnotation) {  
 firstFieldName = constraintAnnotation.first();  
 secondFieldName = constraintAnnotation.second();  
 }  
  
 @Override  
 public boolean isValid(final Object value, final ConstraintValidatorContext context) {  
 try {  
 final Object firstObj = BeanUtils.getProperty(value, firstFieldName);  
 final Object secondObj = BeanUtils.getProperty(value, secondFieldName);  
 return firstObj == null && secondObj == null || firstObj != null && firstObj.equals(secondObj);  
 } catch (final Exception ignore) {}  
 return true;  
 }  
}

**MainController.java**

package prac23.controllers;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
  
@Controller  
public class MainController {  
  
 @GetMapping("/")  
 public String root() {  
 return "index";  
 }  
  
 @GetMapping("/login")  
 public String login(Model model) {  
 return "login";  
 }  
  
 @GetMapping("/user")  
 public String userIndex() {  
 return "user/index";  
 }  
}

**RegistrationController.java**

package prac23.controllers;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.validation.BindingResult;  
import org.springframework.web.bind.annotation.\*;  
import prac23.model.User;  
import prac23.service.UserService;  
  
import javax.validation.Valid;  
  
@Controller  
@RequestMapping("/registration")  
public class RegistrationController {  
  
 @Autowired  
 private UserService userService;  
  
 @ModelAttribute("user")  
 public UserRegistrationDto userRegistrationDto() {  
 return new UserRegistrationDto();  
 }  
  
 @GetMapping  
 public String showRegistrationForm(Model model) {  
 return "registration";  
 }  
  
 @PostMapping  
 public String registerUserAccount(@ModelAttribute("user") @Valid UserRegistrationDto userDto,  
 BindingResult result, @CookieValue("JSESSIONID") String cookie){  
  
 User existing = userService.findByLogin(userDto.getLogin());  
 if (existing != null){  
 result.rejectValue("login", null,  
 "There is already an account registered with that login");  
 }  
  
 if (result.hasErrors()){  
 return "registration";  
 }  
  
 userService.save(userDto, cookie);  
 return "redirect:/registration?success";  
 }  
  
}

**UserRegistrationDto.java**

package prac23.controllers;  
  
import prac23.constraint.FieldMatch;  
  
import javax.validation.constraints.AssertTrue;  
import javax.validation.constraints.NotNull;  
  
@FieldMatch.List({  
 @FieldMatch(first = "password", second = "confirmPassword", message = "The password fields must match"),  
 @FieldMatch(first = "login", second = "confirmLogin", message = "The login fields must match")  
})  
public class UserRegistrationDto {  
  
 @NotNull  
 private String password;  
  
 @NotNull  
 private String confirmPassword;  
  
 @NotNull  
 private String login;  
  
 @NotNull  
 private String confirmLogin;  
  
 @AssertTrue  
 private Boolean terms;  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 public String getConfirmPassword() {  
 return confirmPassword;  
 }  
  
 public void setConfirmPassword(String confirmPassword) {  
 this.confirmPassword = confirmPassword;  
 }  
  
 public String getLogin() {  
 return login;  
 }  
  
 public void setLogin(String login) {  
 this.login = login;  
 }  
  
 public String getConfirmLogin() {  
 return confirmLogin;  
 }  
  
 public void setConfirmLogin(String confirmEmail) {  
 this.confirmLogin = confirmLogin;  
 }  
  
 public Boolean getTerms() {  
 return terms;  
 }  
  
 public void setTerms(Boolean terms) {  
 this.terms = terms;  
 }  
  
}

**Role.java**

package prac23.model;  
  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
  
@Entity  
public class Role {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 private Long id;  
 private String name;  
  
 public Role() {  
 }  
  
 public Role(Long id, String name) {  
 this.id = id;  
 this.name = name;  
 }  
  
 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;  
 }  
  
 @Override  
 public String toString() {  
 return "Role{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 '}';  
 }  
}

**User.java**

package prac23.model;  
  
import lombok.Getter;  
import lombok.Setter;  
  
import javax.persistence.\*;  
import java.util.Collection;  
  
@Entity  
@Table(name = "users")  
@Setter  
@Getter  
public class User {  
 @Id  
 @Column(name = "user\_id")  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
  
 @Column(name = "login")  
 private String login;  
  
 @Column(name = "password")  
 private String password;  
  
 @Column(name = "session\_id")  
 private String sessionID;  
  
  
 @ManyToMany(fetch=FetchType.EAGER)  
 @JoinTable(name = "user\_roles",  
 joinColumns = @JoinColumn(name = "user\_id"),  
 inverseJoinColumns = @JoinColumn(name = "role\_id"))  
 private Collection<Role> roles;  
  
  
 public User() {  
 }  
  
 public User(String login, String password) {  
 this.login = login;  
 this.password = password;  
 }  
  
 public User(String password, Collection<Role> roles) {  
 this.login = login;  
 this.password = password;  
 this.roles = roles;  
 }  
}

**UserRepository.java**

package prac23.repositories;  
  
import prac23.model.User;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface UserRepository extends JpaRepository<User, Long> {  
  
 User findByLogin(String login);  
  
}

**UserService.java**

package prac23.service;  
  
import prac23.model.User;  
import prac23.controllers.UserRegistrationDto;  
import org.springframework.security.core.userdetails.UserDetailsService;  
  
public interface UserService extends UserDetailsService {  
  
 User findByLogin(String login);  
  
 User save(UserRegistrationDto registration, String cookie);  
}

**UserServiceImpl.java**

package prac23.service;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.stereotype.Service;  
import prac23.controllers.UserRegistrationDto;  
import prac23.model.Role;  
import prac23.model.User;  
import prac23.repositories.UserRepository;  
  
import java.util.Collection;  
import java.util.Collections;  
import java.util.stream.Collectors;  
  
@Service  
public class UserServiceImpl implements UserService {  
  
 @Autowired  
 private UserRepository userRepository;  
  
 @Autowired  
 private BCryptPasswordEncoder passwordEncoder;  
  
 public User findByLogin(String login){  
 return userRepository.findByLogin(login);  
 }  
  
 public User save(UserRegistrationDto registration, String cookie){  
 User user = new User();  
 user.setLogin(registration.getLogin());  
 user.setPassword(passwordEncoder.encode(registration.getPassword()));  
 user.setRoles(Collections.singletonList(new Role(1L,"ROLE\_USER")));  
 user.setSessionID(cookie);  
 return userRepository.save(user);  
 }  
  
 @Override  
 public UserDetails loadUserByUsername(String login) throws UsernameNotFoundException {  
 User user = userRepository.findByLogin(login);  
 if (user == null){  
 throw new UsernameNotFoundException("Invalid username or password.");  
 }  
 return new org.springframework.security.core.userdetails.User(user.getLogin(),  
 user.getPassword(),  
 mapRolesToAuthorities(user.getRoles()));  
 }  
  
 private Collection<? extends GrantedAuthority> mapRolesToAuthorities(Collection<Role> roles){  
 return roles.stream()  
 .map(role -> new SimpleGrantedAuthority(role.getName()))  
 .collect(Collectors.toList());  
 }  
}

**main.css**

h1 {  
 color: #78ab46;  
}  
  
footer {  
 position: fixed;  
 height: 50px;  
 bottom: 0;  
 width: 100%;  
 background-color: #ccc  
}  
  
footer p {  
 padding: 15px;  
}  
  
.error-message {  
 color: #a94442;  
}  
  
.error-message:first-letter {  
 text-transform: capitalize;  
}

**index.html**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
</head>  
<body>  
<h1>Прокофьев Сергей ИКБО-03-19 19</h1>  
</body>  
</html>

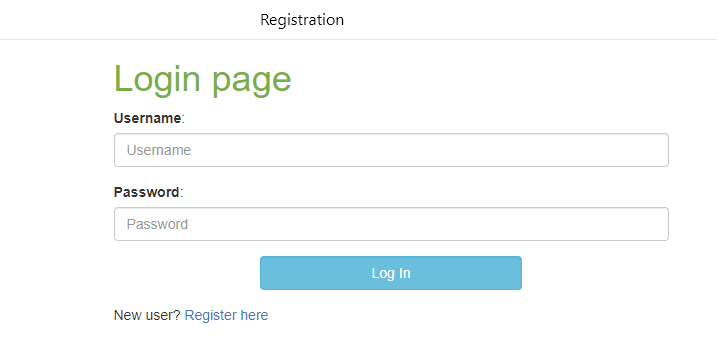
**login.html**

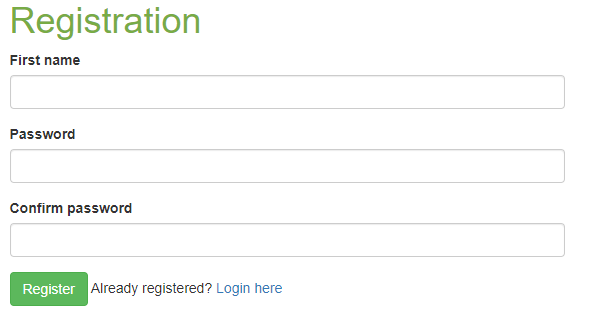
<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="utf-8"/>  
 <meta http-equiv="X-UA-Compatible" content="IE=edge"/>  
 <meta name="viewport" content="width=device-width, initial-scale=1"/>  
  
 <link rel="stylesheet" type="text/css" th:href="@{/webjars/bootstrap/3.3.7/css/bootstrap.min.css}"/>  
 <link rel="stylesheet" type="text/css" th:href="@{/css/main.css}"/>  
  
 <title>Registration</title>  
</head>  
<body>  
<div class="container">  
  
 <div class="row">  
 <div class="col-md-6 col-md-offset-3">  
 <h1>Login page</h1>  
 <form th:action="@{/login}" method="post">  
 <div th:if="${param.error}">  
 <div class="alert alert-danger">  
 Invalid username or password.  
 </div>  
 </div>  
 <div th:if="${param.logout}">  
 <div class="alert alert-info">  
 You have been logged out.  
 </div>  
 </div>  
 <div class="form-group">  
 <label for="username">Username</label>:  
 <input type="text"  
 id="username"  
 name="username"  
 class="form-control"  
 autofocus="autofocus"  
 placeholder="Username"/>  
 </div>  
 <div class="form-group">  
 <label for="password">Password</label>:  
 <input type="password"  
 id="password"  
 name="password"  
 class="form-control"  
 placeholder="Password"/>  
 </div>  
 <div class="form-group">  
 <div class="row">  
 <div class="col-sm-6 col-sm-offset-3">  
 <input type="submit"  
 name="login-submit"  
 id="login-submit"  
 class="form-control btn btn-info"  
 value="Log In"/>  
 </div>  
 </div>  
 </div>  
 <div class="form-group">  
 <span>New user? <a href="/" th:href="@{/registration}">Register here</a></span>  
 </div>  
 </form>  
 </div>  
 </div>  
  
  
</div>  
  
<script type="text/javascript" th:src="@{/webjars/jquery/3.2.1/jquery.min.js/}"></script>  
<script type="text/javascript" th:src="@{/webjars/bootstrap/3.3.7/js/bootstrap.min.js}"></script>  
  
</body>  
</html>

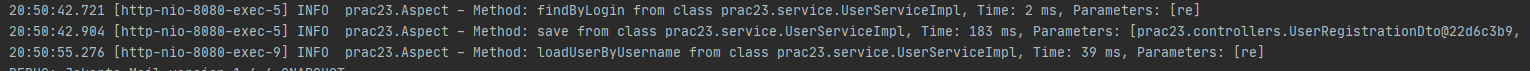
**registration.html**

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="utf-8"/>  
 <meta http-equiv="X-UA-Compatible" content="IE=edge"/>  
 <meta name="viewport" content="width=device-width, initial-scale=1"/>  
  
 <link rel="stylesheet" type="text/css" th:href="@{/webjars/bootstrap/3.3.7/css/bootstrap.min.css}"/>  
 <link rel="stylesheet" type="text/css" th:href="@{/css/main.css}"/>  
  
 <title>Registration</title>  
</head>  
<body>  
<div class="container">  
 <div class="row">  
 <div class="col-md-6 col-md-offset-3">  
  
 <div th:if="${param.success}">  
 <div class="alert alert-info">  
 You've successfully registered to our awesome app!  
 </div>  
 </div>  
  
 <h1>Registration</h1>  
 <form th:action="@{/registration}" th:object="${user}" method="post">  
  
 <p class="error-message"  
 th:if="${#fields.hasGlobalErrors()}"  
 th:each="error : ${#fields.errors('global')}"  
 th:text="${error}">Validation error</p>  
  
 <div class="form-group"  
 th:classappend="${#fields.hasErrors('login')}? 'has-error':''">  
 <label for="login" class="control-label">First name</label>  
 <input id="login"  
 class="form-control"  
 th:field="\*{login}"/>  
 <p class="error-message"  
 th:each="error: ${#fields.errors('login')}"  
 th:text="${error}">Validation error</p>  
 </div>  
 <div class="form-group"  
 th:classappend="${#fields.hasErrors('password')}? 'has-error':''">  
 <label for="password" class="control-label">Password</label>  
 <input id="password"  
 class="form-control"  
 type="password"  
 th:field="\*{password}"/>  
 <p class="error-message"  
 th:each="error : ${#fields.errors('password')}"  
 th:text="${error}">Validation error</p>  
 </div>  
 <div class="form-group"  
 th:classappend="${#fields.hasErrors('confirmPassword')}? 'has-error':''">  
 <label for="confirmPassword" class="control-label">Confirm password</label>  
 <input id="confirmPassword"  
 class="form-control"  
 type="password"  
 th:field="\*{confirmPassword}"/>  
 <p class="error-message"  
 th:each="error : ${#fields.errors('confirmPassword')}"  
 th:text="${error}">Validation error</p>  
 </div>  
 <div class="form-group">  
 <button type="submit" class="btn btn-success">Register</button>  
 <span>Already registered? <a href="/" th:href="@{/login}">Login here</a></span>  
 </div>  
  
 </form>  
 </div>  
 </div>  
</div>  
  
<script type="text/javascript" th:src="@{/webjars/jquery/3.2.1/jquery.min.js/}"></script>  
<script type="text/javascript" th:src="@{/webjars/bootstrap/3.3.7/js/bootstrap.min.js}"></script>  
  
</body>  
</html>

**Результаты:**

****

****

****

**Выводы:**

Проделав данную работу, мы добавили защищённую регистрацию и вход для пользователей через Spring Security, защиту всех запросов, кроме запросов на авторизацию и регистрацию, от неавторизированных пользователей.

# Практическая работа №24

**Задание:** Написать модульное тестирование для всех классов сервисов приложения из предыдущего задания.

**Цель работы:** Тестирование в Spring Framework с использованием Junit.

**Выполнение работы:**

**Testing24.java**

package prac24;  
  
import org.junit.jupiter.api.Test;  
import org.springframework.boot.test.context.SpringBootTest;  
  
@SpringBootTest  
public class Testing24 {  
 @Test  
 void contextLoads() {  
 }  
}

**DepartureTest.java**

package prac24.service;  
  
  
import org.junit.Before;  
import org.junit.jupiter.api.Assertions;  
import org.junit.jupiter.api.Test;  
import org.junit.runner.RunWith;  
import org.mockito.ArgumentCaptor;  
import org.mockito.Captor;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.boot.test.mock.mockito.MockBean;  
import org.springframework.test.context.junit4.SpringRunner;  
import prac24.model.Departure;  
import prac24.repositories.DepartureRepository;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
import static org.mockito.Mockito.\*;  
  
@SpringBootTest  
@RunWith(SpringRunner.class)  
public class DepartureTest {  
 @MockBean  
 private DepartureRepository departureRepository;  
  
 @Captor  
 private ArgumentCaptor<Departure> captor;  
 private DepartureService departureService;  
  
 @Autowired  
 public void setDepartureService(DepartureService departureService) {  
 this.departureService = departureService;  
 }  
  
 private final List<Departure> departures = new ArrayList<>();  
  
 @Before  
 public void setUp() {  
 Departure departure1 = new Departure();  
 departure1.setId(0);  
 departure1.setDepartureDate("13/09/2021");  
 departure1.setPostoffice\_id(1);  
 departure1.setType("Hm");  
  
 Departure departure2 = new Departure();  
 departure2.setId(1);  
 departure2.setDepartureDate("12/09/2021");  
 departure2.setPostoffice\_id(1);  
 departure2.setType("nehm");  
  
 departures.add(departure1);  
 departures.add(departure2);  
  
 when(departureRepository.findAll())  
 .thenReturn(departures);  
 }  
  
 @Test  
 public void whenGetDepartures() {  
 List<Departure> fetched = departureService.getDepartures();  
  
 verify(departureRepository, times(1))  
 .findAll();  
 Assertions.assertEquals(departures.size(), fetched.size());  
 }  
  
 @Test  
 public void addDeparture() {  
 Departure departure = new Departure();  
 departure.setId(1);  
 departure.setDepartureDate("13/09/2021");  
 departure.setPostoffice\_id(1);  
 departure.setType("Hm");  
 departureService.addDeparture(departure);  
  
 verify(departureRepository).save(captor.capture());  
 Departure captured = captor.getValue();  
 Assertions.assertEquals(departure.getDepartureDate(), captured.getDepartureDate());  
 Assertions.assertEquals(departure.getId(), captured.getId());  
 Assertions.assertEquals(departure.getType(), captured.getType());  
 Assertions.assertEquals(departure.getPostoffice\_id(), captured.getPostoffice\_id());  
 }  
  
 @Test  
 public void whenDelete() {  
 Departure departure = new Departure();  
 departure.setId(1);  
 departure.setDepartureDate("13/09/2021");  
 departure.setPostoffice\_id(1);  
 departure.setType("Hm");  
 departureService.addDeparture(departure);  
  
 verify(departureRepository).save(captor.capture());  
 Departure captured = captor.getValue();  
 System.out.println(captured.getId());  
  
 when(departureRepository.findById(captured.getId()))  
 .thenReturn(Optional.of(captured));  
 when(departureRepository.existsById(captured.getId()))  
 .thenReturn(true);  
  
 departureService.deleteDeparture(captured.getId());  
  
 Assertions.assertEquals(departureService.getDepartures().size(), 0);  
 }  
}

**EmailTest.java**

package prac24.service;  
  
import org.junit.jupiter.api.Test;  
import org.junit.runner.RunWith;  
import org.mockito.ArgumentCaptor;  
import org.mockito.ArgumentMatchers;  
import org.mockito.Captor;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.boot.test.mock.mockito.MockBean;  
import org.springframework.mail.SimpleMailMessage;  
import org.springframework.mail.javamail.JavaMailSender;  
import org.springframework.test.context.junit4.SpringRunner;  
  
import static org.junit.Assert.assertEquals;  
import static org.mockito.Mockito.times;  
import static org.mockito.Mockito.verify;  
  
@SpringBootTest  
@RunWith(SpringRunner.class)  
public class EmailTest {  
 private EmailService emailService;  
 @MockBean  
 private JavaMailSender emailSender;  
 @Captor  
 private ArgumentCaptor<SimpleMailMessage> captor;  
  
 @Autowired  
 public void setEmailService(EmailService emailService) {  
 this.emailService = emailService;  
 }  
  
 @Test  
 public void sendValidEmail() {  
 String subject = "Test";  
 String text = "Text";  
 emailService.sendMail(subject, text);  
  
 verify(emailSender, times(1))  
 .send(ArgumentMatchers.any(SimpleMailMessage.class));  
 verify(emailSender, times(1))  
 .send(captor.capture());  
  
 SimpleMailMessage actual = captor.getValue();  
 assertEquals(subject, actual.getSubject());  
 assertEquals(text, actual.getText());  
 }  
}

**PostOfficeTest.java**

package prac24.service;  
  
import org.junit.Before;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.Assertions;  
import org.junit.runner.RunWith;  
import org.mockito.ArgumentCaptor;  
import org.mockito.Captor;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.boot.test.mock.mockito.MockBean;  
import org.springframework.test.context.junit4.SpringRunner;  
import prac24.model.PostOffice;  
import prac24.repositories.PostOfficeRepository;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
import static org.mockito.Mockito.\*;  
  
  
@SpringBootTest  
@RunWith(SpringRunner.class)  
public class PostOfficeTest {  
 @MockBean  
 private PostOfficeRepository postOfficeRepository;  
  
 @Captor  
 private ArgumentCaptor<PostOffice> captor;  
 private PostOfficeService postOfficeService;  
  
 @Autowired  
 public void setPostOfficeService(PostOfficeService postOfficeService) {  
 this.postOfficeService = postOfficeService;  
 }  
 private final List<PostOffice> posts = new ArrayList<>();  
  
 @Before  
 public void setUp(){  
 PostOffice post1 =new PostOffice();  
 post1.setId(0);  
 post1.setCityName("Moscow");  
 post1.setName("Kek");  
  
 PostOffice post2 =new PostOffice();  
 post2.setId(1);  
 post2.setCityName("SPB");  
 post2.setName("Hello");  
  
 posts.add(post1);  
 posts.add(post2);  
  
 when(postOfficeRepository.findAll()).thenReturn(posts);  
 }  
  
 @Test  
 public void whenGetPostOffices(){  
 List<PostOffice> fetched = postOfficeService.getPostOffices();  
  
 verify(postOfficeRepository, times(1)).findAll();  
 Assertions.assertEquals(posts.size(), fetched.size());  
 }  
  
 @Test  
 public void addPostOffice(){  
 PostOffice post3 =new PostOffice();  
 post3.setId(3);  
 post3.setCityName("Mos");  
 post3.setName("Name");  
 postOfficeService.addPostOffice(post3);  
  
 verify(postOfficeRepository).save(captor.capture());  
 PostOffice captured = captor.getValue();  
 Assertions.assertEquals(post3.getName(), captured.getName());  
 Assertions.assertEquals(post3.getCityName(), captured.getCityName());  
 Assertions.assertEquals(post3.getId(), captured.getId());  
 }  
  
 @Test  
 public void whenDelete() {  
 PostOffice post4 =new PostOffice();  
 post4.setId(3);  
 post4.setCityName("Rim");  
 post4.setName("PePe");  
 postOfficeService.addPostOffice(post4);  
  
 verify(postOfficeRepository).save(captor.capture());  
 PostOffice captured = captor.getValue();  
 System.out.println(captured.getId());  
  
 when(postOfficeRepository.findById(captured.getId()))  
 .thenReturn(Optional.of(captured));  
 when(postOfficeRepository.existsById(captured.getId()))  
 .thenReturn(true);  
  
 postOfficeService.deletePostOffice(captured.getId());  
  
 Assertions.assertEquals(postOfficeService.getPostOffices().size(), 0);  
 }  
}

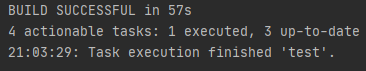
**SchedulerTest.java**

package prac24.service;  
  
import org.junit.jupiter.api.Test;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.boot.test.mock.mockito.MockBean;  
import org.springframework.test.context.junit4.SpringRunner;  
import org.springframework.util.ResourceUtils;  
  
import java.io.File;  
import java.io.IOException;  
import java.util.Arrays;  
import java.util.Objects;  
import java.util.stream.Collectors;  
  
import static org.junit.Assert.assertTrue;  
import static org.mockito.Mockito.times;  
import static org.mockito.Mockito.verify;  
  
@SpringBootTest  
@RunWith(SpringRunner.class)  
public class SchedulerTest {  
 private SchedulerService schedulerService;  
 @MockBean  
 private PostOfficeService postOfficeService;  
 @MockBean  
 private DepartureService departureService;  
  
 @Autowired  
 public void setDataScheduler(SchedulerService schedulerService) {  
 this.schedulerService = schedulerService;  
 }  
  
 @Test  
 public void checkDataFilesExist() throws IOException {  
 schedulerService.doScheduledTask();  
  
 String path = "src/main/resources/out";  
 File dir = ResourceUtils.getFile(path);  
 verify(postOfficeService, times(1)).getPostOffices();  
 verify(departureService, times(1)).getDepartures();  
 assertTrue(dir.exists());  
 String[] files = {"departure.txt", "postOffice.txt"};  
  
 assertTrue(Arrays.stream(Objects.requireNonNull(dir.listFiles()))  
 .map(File::getName)  
 .collect(Collectors.toList())  
 .containsAll(Arrays.asList(files)));  
 }  
}

**UserTest.java**

package prac24.service;  
  
import org.junit.Before;  
import org.junit.jupiter.api.Test;  
import org.junit.runner.RunWith;  
import org.mockito.ArgumentCaptor;  
import org.mockito.Captor;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.boot.test.mock.mockito.MockBean;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.test.context.junit4.SpringRunner;  
import prac24.model.User;  
import prac24.repositories.UserRepository;  
  
import static org.assertj.core.api.Assertions.assertThat;  
import static org.junit.Assert.assertEquals;  
import static org.mockito.ArgumentMatchers.any;  
import static org.mockito.Mockito.verify;  
import static org.mockito.Mockito.when;  
  
  
@SpringBootTest  
@RunWith(SpringRunner.class)  
public class UserTest {  
 @MockBean  
 private UserRepository userRepository;  
 private UserService userService;  
 private PasswordEncoder passwordEncoder;  
  
 @Captor  
 private ArgumentCaptor<User> captor;  
  
 @Autowired  
 public void setUserAuthService(UserService userService) {  
 this.userService = userService;  
 }  
  
 @Autowired  
 public void setPasswordEncoder(PasswordEncoder passwordEncoder) {  
 this.passwordEncoder = passwordEncoder;  
 }  
 @Before  
 public void setUp() {  
 User user = new User();  
 user.setLogin("user");  
 user.setPassword("password");  
 user.setId(1L);  
  
 when(userRepository.findByLogin(user.getLogin())).thenReturn(user);  
 }  
 @Test  
 public void UserShouldBeRegister() {  
 String encodedPassword = passwordEncoder.encode("qwerty");  
 User newUser = new User();  
 newUser.setLogin("test");  
 newUser.setPassword(encodedPassword);  
 newUser.setId(2L);  
  
 when(userRepository.save(any(User.class))).thenReturn(newUser);  
 User user = userRepository.save(newUser);  
  
 assertThat(user).isNotNull();  
 verify(userRepository).save(captor.capture());  
 User captured = captor.getValue();  
 assertEquals(newUser.getLogin(), captured.getLogin());  
 }  
}

**Результаты:**

****

****

**Выводы:** В результате данной работы написано тестирование и тестировочные тесты для проверки работоспособности данного пректа.