|  |
| --- |
|  |
| 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 № 1 — 24** | |
| **по дисциплине** | |
| **«Шаблоны программных платформ языка Джава»** | |
| Выполнил студент группы ИКБО-36-22 | ***Кужугет С.А.*** |
| Принял ассистент кафедры ИиППО | ***Ермаков С.Р.*** |

|  |  |  |
| --- | --- | --- |
| Практические работы выполнены | « » 2024г. |  |
| «Зачтено» | « » 2024г. |  |

Москва 2024

**Практическая работа №1**

1. **Постановка задачи и персональный вариант**

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

Постановка задачи: Имплементировать интерфейс Comparator, сравнивающий два числа по модулю.

1. **Код и пояснение к коду**

Листинг 1 – код для пр. работы №1

package task1;  
  
import java.util.Comparator;  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Comparator<Integer> comparator = (a, b) -> Integer.*compare*(Math.*abs*(a), Math.*abs*(b));  
 Scanner input = new Scanner(System.*in*);  
 int value1 = input.nextInt();  
 int value2 = input.nextInt();  
 if (comparator.compare(value1, value2) == 0){  
 System.*out*.println("Equal!");  
 }  
 else{  
 System.*out*.println("Not equal!");  
 }  
 }  
}

1. **Результат работы программы**

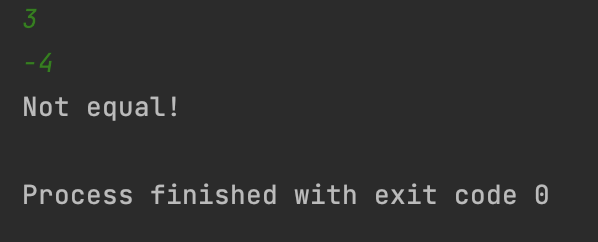
****

Рисунок 1 – Результат работы программы

**Практическая работа №2**

1. **Постановка задачи и персональный вариант**

Цель работы: Работа со Stream API в Java 8.

Постановка задачи: Выбор первых 5 элементов списка, сортировка по дате рождения от старых к новым, фильтрация по весу меньше, чем 60, вывод имени и фамилии через пробел.

1. **Код и пояснение к коду**

Листинг 2 – код для пр. работы №2

package task2;  
  
import java.time.LocalDate;  
  
public record Human(int age, String firstName, String lastName, LocalDate birthDate, int weight){  
}

Листинг 3 – код для пр. работы №2

package task2;  
import java.util.\*;  
import java.time.LocalDate;  
  
public class Main {  
 public static void main(String[] args){  
 List<Human> humans = Arrays.*asList*(  
 new Human(20, "John", "Doe", LocalDate.*of*(2001, 1, 1), 70),  
 new Human(30, "Jane", "Bosco", LocalDate.*of*(1991, 2, 2), 50),  
 new Human(25, "Tom", "Smith", LocalDate.*of*(1996, 3, 3), 56),  
 new Human(35, "Alice", "Brown", LocalDate.*of*(1986, 4, 13), 80),  
 new Human(66, "Joshua", "Hill", LocalDate.*of*(1999, 5, 14), 50),  
 new Human(34, "Matthew", "Sanders", LocalDate.*of*(2005, 5, 13), 73),  
 new Human(19, "Bob", "Peterson", LocalDate.*of*(1994, 2, 10), 60)  
 );  
 List <Human> stream = humans.stream()  
 .limit(5)  
 .sorted((human1, human2) -> human2.birthDate().compareTo(human1.birthDate()))  
 .filter(human -> human.weight() < 60).toList();  
 for(Human human : stream){  
 System.*out*.println(human.firstName() + " " + human.lastName());  
 }  
 }  
}

1. **Результат работы программы**

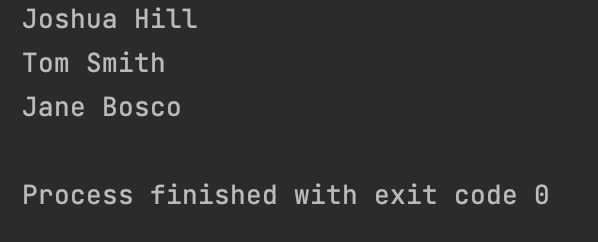
****

Рисунок 2 – Результат работы программы

**Практическая работа №3**

1. **Постановка задачи и персональный вариант**

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

Постановка задачи: List с использованием ключевого слова synchronized, Set с использованием Semaphore.

1. **Код и пояснение к коду**

Листинг 4 – код для пр. работы №3

package task3;  
  
import java.util.\*;  
import java.util.concurrent.Semaphore;  
  
//List с использованием ключевого слова synchronized, Set с использованием Semaphore.  
public class Main {  
 static List<Integer> *integerList* = new ArrayList<>();  
 static Set<Integer> *integerSet* = new HashSet<>();  
 synchronized static void incrementList(int i) {  
 *integerList*.add(i);  
 }  
 private static final Semaphore *semaphore* = new Semaphore(1);  
 static void incrementSet(int i) {  
 try {  
 *semaphore*.acquire();  
 *integerSet*.add(i);  
 *semaphore*.release();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 } }  
 public static void main(String[] args) throws Exception {  
 Thread listOne = new Thread(() -> {  
 for (int i = 0; i < 50; i++) {  
 *incrementList*(i);  
 }  
 });  
 Thread listTwo = new Thread(() -> {  
 for (int i = 0; i < 50; i++) {  
 *incrementList*(i);  
 }  
 });  
 listOne.start();  
 listTwo.start();  
 Thread.*sleep*(3000);  
 System.*out*.println(*integerList*);  
 System.*out*.println("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");  
 Thread setOne = new Thread(()->{  
 for (int i = 0; i < 50; i++) {  
 *incrementSet*(i);  
 }  
 });  
 Thread setTwo = new Thread(()->{  
 for (int i = 50; i < 99; i++) {  
 *incrementSet*(i);  
 } });  
 setTwo.start();  
 setOne.start();  
 Thread.*sleep*(3000);  
 System.*out*.println(*integerSet*);  
 }  
}

1. **Результат работы программы**

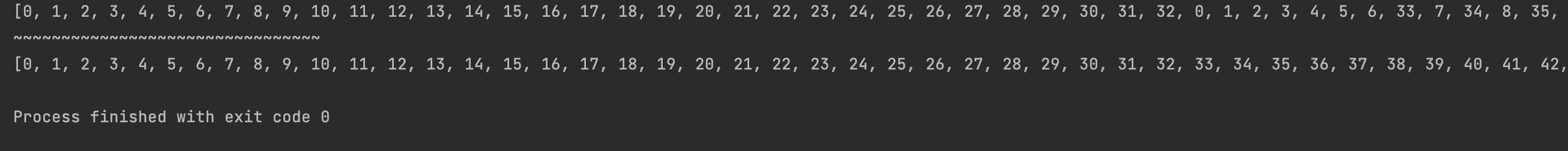
****

Рисунок 3 – Результат работы программы

**Практическая работа №4**

1. **Постановка задачи и персональный вариант**

Цель работы: Работа с ExecutorService, CompletableFuture.

Постановка задачи: Реализовать собственную имплементацию ExecutorService с единственным параметром конструктора – количеством потоков.

1. **Код и пояснение к коду**

Листинг 5 – код для пр. работы №4

package task4;  
  
public class Main {  
 public static void main(String[] args) {  
 ExecutorServiceClass service = new ExecutorServiceClass(1);  
 service.execute(() -> {  
 try{  
 Thread.*sleep*(200);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 System.*out*.println("We run it");  
 }  
 );  
 service.submit(() -> System.*out*.println("Start"));  
 }  
}

Листинг 6 – код для пр. работы №4 класс ExecutorServiceClass

package com.company.practics.practic\_4;  
package task4;  
  
import java.util.Collection;  
import java.util.List;  
import java.util.concurrent.\*;  
  
public class ExecutorServiceClass implements ExecutorService{  
 ExecutorService executorService;  
  
 public ExecutorServiceClass(int threads){  
 executorService = Executors.*newFixedThreadPool*(threads);  
 }  
 @Override  
 public void shutdown() {  
 executorService.shutdown();  
 }  
  
 @Override  
 public List<Runnable> shutdownNow() {  
 return executorService.shutdownNow();  
 }  
  
 @Override  
 public boolean isShutdown() {  
 return executorService.isShutdown();  
 }  
 @Override  
 public boolean isTerminated() {  
 return executorService.isTerminated();  
 }  
  
 @Override  
 public boolean awaitTermination(long timeout, TimeUnit unit) throws InterruptedException {  
 return executorService.awaitTermination(timeout, unit);  
 }  
  
 @Override  
 public <T> Future<T> submit(Callable<T> task) {  
 return executorService.submit(task);  
 }  
  
 @Override  
 public <T> Future<T> submit(Runnable task, T result) {  
 return executorService.submit(task, result);  
 }  
  
 @Override  
 public Future<?> submit(Runnable task) {  
 return executorService.submit(task);  
 }  
  
 @Override  
 public <T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks) throws InterruptedException {  
 return executorService.invokeAll(tasks);  
 }  
  
 @Override  
 public <T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks, long timeout, TimeUnit unit) throws InterruptedException {  
 return executorService.invokeAll(tasks, timeout, unit);  
 }  
  
 @Override  
 public <T> T invokeAny(Collection<? extends Callable<T>> tasks) throws InterruptedException, ExecutionException {  
 return executorService.invokeAny(tasks);  
 }  
  
 @Override  
 public <T> T invokeAny(Collection<? extends Callable<T>> tasks, long timeout, TimeUnit unit) throws InterruptedException, ExecutionException, TimeoutException {  
 return invokeAny(tasks, timeout, unit);  
 }  
  
 @Override  
 public void execute(Runnable command) {  
 executorService.execute(command);  
 }  
}

1. **Результат работы программы**

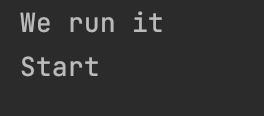
****

Рисунок 4 – Результат работы программы

**Практическая работа №5**

1. **Постановка задачи и персональный вариант**

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

Постановка задачи: Реализовать паттерн Singleton как минимум 3-мя способами.

1. **Код и пояснение к коду**

Листинг 7 – код для пр. работы №5

package task5;  
  
public class Main {  
 public static void main(String[] args) {  
 Singleton1 singleton1 = Singleton1.*getInstance*();  
 Singleton1 singleton11 = Singleton1.*getInstance*();  
 System.*out*.print("Equality singleton 1: ");  
 System.*out*.println(singleton11.equals(singleton1));  
  
 Singleton2 singleton2 = Singleton2.*getInstance*();  
 Singleton2 singleton22 = Singleton2.*getInstance*();  
 System.*out*.print("Equality singleton 2: ");  
 System.*out*.println(singleton22.equals(singleton2));  
  
 Singleton3 singleton3 = Singleton3.*getInstance*();  
 Singleton3 singleton33 = Singleton3.*getInstance*();  
 System.*out*.print("Equality singleton 3: ");  
 System.*out*.println(singleton33.equals(singleton3));  
 }  
}

Листинг 8 – код для пр. работы №5 класс Singleton1

package task5;  
  
public class Singleton1 {  
 private static Singleton1 *instance*;  
  
 private Singleton1(){}  
  
 public static synchronized Singleton1 getInstance() {  
 if (*instance* == null){  
 *instance* = new Singleton1();  
 }  
 return *instance*;  
 }  
}

Листинг 9 – код для пр. работы №5 класс Singleton2

package task5;  
  
public class Singleton2 {  
 private static Singleton2 *instance* = new Singleton2();  
  
 private Singleton2(){}  
  
 public static Singleton2 getInstance(){  
 return *instance*;  
 }  
}

Листинг 10 – код для пр. работы №5 класс Singleton3

package task5;  
  
public enum Singleton3 {  
 *INSTANCE*;  
  
 public static Singleton3 getInstance(){  
 return *INSTANCE*;  
 }  
}

1. **Результат работы программы**

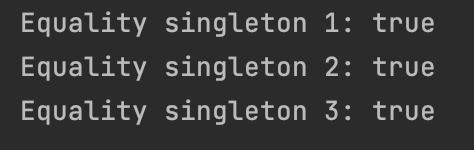
****

Рисунок 5 – Результат работы программы

**Практическая работа №6**

1. **Постановка задачи и персональный вариант**

Цель работы: Знакомство с реализацией порождающих паттернов проектирования.

Постановка задачи: Написать реализацию паттернов «Фабричный метод», «Абстрактная фабрика», «Строитель», «Прототип».

1. **Код и пояснение к коду**

Листинг 11 – код для пр. работы №6

package task6.abstract\_factory;  
  
public class Main {  
 public static void main(String[] args) {  
 IceCreamFactory factoryA = new IceCreamFactoryA();  
 IceCreamFactoryB factoryB = new IceCreamFactoryB();  
  
 factoryA.createGelato().print();  
 factoryA.createSorbet().print();  
  
 factoryB.createGelato().print();  
 factoryB.createSorbet().print();  
 }  
}

Листинг 12 – код для пр. работы №6 класс IceCreamFactoryA

package task6.abstract\_factory;  
  
public class IceCreamFactoryA implements IceCreamFactory{  
 @Override  
 public Gelato createGelato() {  
 return new GelatoA();  
 }  
  
 @Override  
 public Sorbet createSorbet() {  
 return new SorbetA();  
 }  
}

Листинг 13 – код для пр. работы №6 класс IceCreamFactoryB

package task6.abstract\_factory;  
  
public class IceCreamFactoryB implements IceCreamFactory{  
 @Override  
 public Sorbet createSorbet() {  
 return new SorbetB();  
 }  
  
 @Override  
 public Gelato createGelato() {  
 return new GelatoB();  
 }  
}

Листинг 14 – код для пр. работы №6 интерфейс IceCreamFactory

package task6.abstract\_factory;  
  
public interface IceCreamFactory {  
 Sorbet createSorbet();  
 Gelato createGelato();  
}

Листинг 15 – код для пр. работы №6

package task6.builder;  
  
public class Main {  
 public static void main(String[] args) {  
 Car chaser = new Car.CarBuilder()  
 .setBrand("Toyota")  
 .setModel("Chaser")  
 .setYear(1999)  
 .setColor("White")  
 .build();  
 System.*out*.println(chaser);  
  
 Car lancer = new Car.CarBuilder()  
 .setBrand("Mitsubishi")  
 .setModel("Lancer Evolution")  
 .setYear(1999)  
 .setColor("Red")  
 .build();  
 System.*out*.println(lancer);  
 }  
}

Листинг 16 – код для пр. работы №6 класс Car

package task6.builder;  
  
public class Car{  
 private String brand;  
 private String model;  
 private int year;  
 private String color;  
  
 public Car(String brand, String model, int year, String color){  
 this.brand = brand;  
 this.model = model;  
 this.year = year;  
 this.color = color;  
 }  
 public static class CarBuilder{  
 private String brand;  
 private String model;  
 private int year;  
 private String color;  
  
 public CarBuilder setBrand(String brand) {  
 this.brand = brand;  
 return this;  
 }  
 public CarBuilder setModel(String model) {  
 this.model = model;  
 return this;  
 }  
 public CarBuilder setYear(int year) {  
 this.year = year;  
 return this;  
 }  
 public CarBuilder setColor(String color) {  
 this.color = color;  
 return this;  
 }  
 public Car build(){  
 return new Car(brand, model, year, color);  
 }  
 }  
  
  
 @Override  
 public String toString() {  
 return "Car{" +  
 "brand: " + this.brand + " | " +  
 "model: " + this.model + " | " +  
 "year: " + this.year + " | " +  
 "color: " + this.color + "}";  
 }  
}

Листинг 17 – код для пр. работы №6

package task6.factory;  
  
public class Main {  
 public static void main(String[] args) {  
 Vehicle car = VehicleFactory.*createVehicle*("car");  
 car.start();  
 car.stop();  
  
 Vehicle motorcycle = VehicleFactory.*createVehicle*("motorcycle");  
 motorcycle.start();  
 motorcycle.stop();  
 }  
}

Листинг 18 – код для пр. работы №6 класс VehicleFactory

package task6.factory;  
  
public class VehicleFactory {  
 public static Vehicle createVehicle(String type){  
 switch(type){  
 case "car":  
 return new Car();  
 case "motorcycle":  
 return new Motorcycle();  
 default:  
 throw new IllegalArgumentException("Invalid vehicle type.");  
 }  
 }  
}

Листинг 19 – код для пр. работы №6

package task6.prototype;  
  
public class Main {  
 public static void main(String[] args) {  
 Square square = new Square();  
 Triangle triangle = new Triangle();  
  
 Shape cloneSquare = square.clone();  
 Shape cloneTriangle = triangle.clone();  
  
 System.*out*.println(cloneTriangle.getType());  
 cloneTriangle.draw();  
  
 System.*out*.println(cloneSquare.getType());  
 cloneSquare.draw();  
 }  
}

Листинг 20 – код для пр. работы №6 класс Shape

package task6.prototype;  
  
public abstract class Shape implements Cloneable{  
 private String type;  
  
 public String getType() {  
 return type;  
 }  
  
 public void setType(String type) {  
 this.type = type;  
 }  
  
 public abstract void draw();  
  
 @Override  
 public Shape clone() {  
 try {  
 return (Shape) super.clone();  
 } catch (CloneNotSupportedException e) {  
 throw new AssertionError();  
 }  
 }  
}

Листинг 21 – код для пр. работы №6 класс Square

package task6.prototype;  
  
public class Square extends Shape{  
 public Square(){  
 setType("square");  
 }  
  
 @Override  
 public void draw() {  
 System.*out*.println("Draw a square.");  
 }  
}

Листинг 22 – код для пр. работы №6 класс Triangle

package task6.prototype;  
  
public class Triangle extends Shape{  
 public Triangle(){  
 setType("triangle");  
 }  
  
 @Override  
 public void draw() {  
 System.*out*.println("Draw a triangle.");  
 }  
}

1. **Результат работы программы**

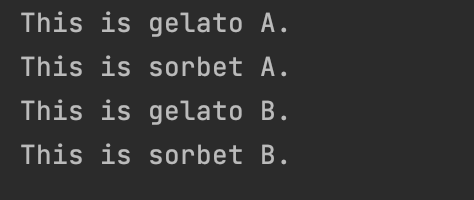
****

Рисунок 6 – Результат работы программы

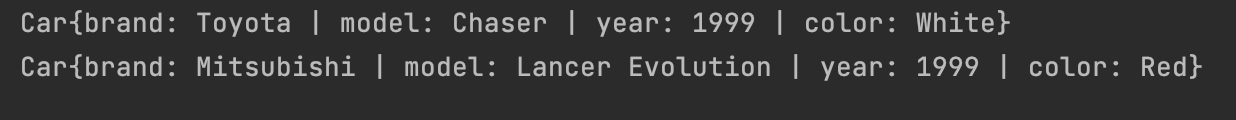


Рисунок 7 – Результат работы программы

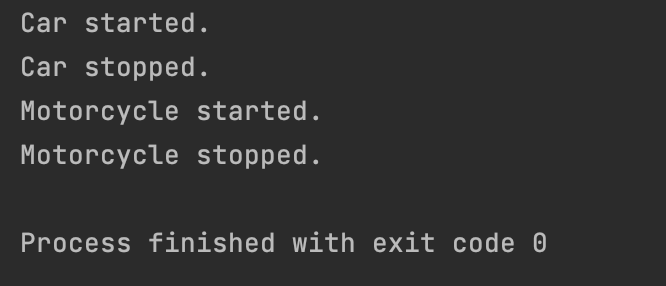


Рисунок 8 – Результат работы программы

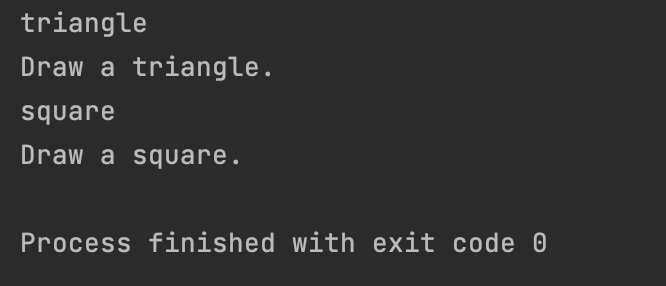


Рисунок 9 – Результат работы программы

**Практическая работа №7**

1. **Постановка задачи и персональный вариант**

Цель работы: Реализация структурных паттернов проектирования.

Постановка задачи: Написать реализацию паттерна в соответствии с вариантом индивидуального задания. 7) Заместитель, Адаптер.

1. **Код и пояснение к коду**

Листинг 23 – код для пр. работы №7

package task7.Adapter;  
  
public class Main {  
 public static void main(String[] args) {  
 Target Adapter = new Adapter(new Adaptee());  
  
 Adapter.request();  
 }  
}

Листинг 24– код для пр. работы №7 класс Adapter

package task7.Adapter;  
  
public class Adapter implements Target{  
 private Adaptee adaptee;  
  
 public Adapter(Adaptee adaptee){  
 this.adaptee = adaptee;  
 }  
  
 @Override  
 public void request() {  
 adaptee.specificRequest();  
 }  
}

Листинг 25 – код для пр. работы №7 класс Adaptee

package task7.Adapter;  
  
public class Adaptee {  
 public void specificRequest(){  
 System.*out*.println("This is adaptee.");  
 }  
}

Листинг 26 – код для пр. работы №7 класс Adaptee

package task7.Adapter;  
  
public interface Target {  
 void request();  
}

Листинг 27 – код для пр. работы №7

package task7.Proxy;  
  
public class Main {  
 public static void main(String[] args) {  
 Subject subject = new Proxy();  
 subject.request();  
 }  
}

Листинг 28 – код для пр. работы №7 класс RealSubject

package task7.Proxy;  
  
public class RealSubject implements Subject{  
 @Override  
 public void request() {  
 System.*out*.println("This is real subject's request.");  
 }  
}

Листинг 29– код для пр. работы №7 класс Proxy

package task7.Proxy;  
  
public class Proxy implements Subject{  
 private RealSubject realSubject;  
  
 @Override  
 public void request() {  
 if (realSubject == null){  
 realSubject = new RealSubject();  
 }  
 realSubject.request();  
 }  
}

Листинг 30 – код для пр. работы №7 интерфейс Subject

package task7.Proxy;  
  
public interface Subject {  
 void request();  
}

1. **Результат работы программы**

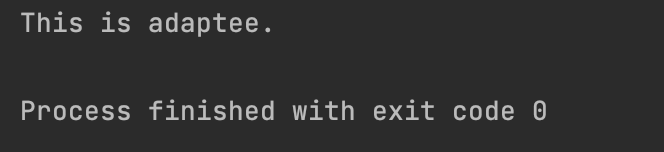
****

Рисунок 10 – Результат работы программы

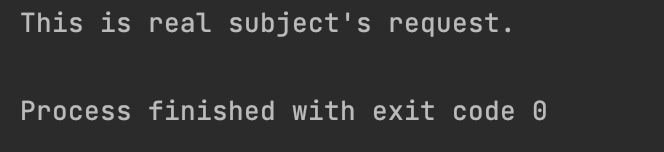


Рисунок 11 – Результат работы программы

**Практическая работа №8**

1. **Постановка задачи и персональный вариант**

Цель работы: Реализация поведенческих паттернов проектирования.

Постановка задачи: Написать реализацию паттерна в соответствии с вариантом индивидуального задания. Стратегия (позволяет определить семейство различных алгоритмов, которые можно заменять), Шаблонный (можно задать некоторое поведение в методе абстрактного класса, но при этом оставить детали реализации на классы-наследники) метод.

1. **Код и пояснение к коду**

Листинг 31 – код для пр. работы №8

package task8.Strategy;  
  
public class Main {  
 public static void main(String[] args) {  
 ElectricCar electricCar = new ElectricCar();  
 GasCar gasCar = new GasCar();  
  
 electricCar.fill();  
 gasCar.fill();  
 }  
}

Листинг 32 – код для пр. работы №8 интерфейс FillStrategy

package task8.Strategy;  
  
public interface FillStrategy {  
 void fill();  
}

Листинг 33 – код для пр. работы №8 класс FillGasStrategy

package task8.Strategy;  
  
public class FillGasStrategy implements FillStrategy {  
 @Override  
 public void fill() {  
 System.*out*.println("Fill your car with gas.");  
 }  
}

Листинг 34 – код для пр. работы №8 класс FillElectricityStrategy

package task8.Strategy;  
  
public class FillElectricityStrategy implements FillStrategy {  
 @Override  
 public void fill() {  
 System.*out*.println("Charge your car.");  
 }  
}

Листинг 35 – код для пр. работы №8 класс ElectricCar

package task8.Strategy;  
  
public class ElectricCar extends Car{  
  
 ElectricCar(){  
 this.fillStrategy = new FillElectricityStrategy();  
 }  
}

Листинг 36 – код для пр. работы №8 класс GasCar

package task8.Strategy;  
  
public class GasCar extends Car{  
  
 GasCar(){  
 this.fillStrategy = new FillGasStrategy();  
 }  
}

Листинг 37 – код для пр. работы №8

package task8.TemplateMethod;  
  
public class Main {  
 public static void main(String[] args) {  
 Document HTMLDocument = new HTMLDocument();  
 HTMLDocument.createDocument();  
 }  
}

Листинг 38 – код для пр. работы №8 класс Document

package task8.TemplateMethod;  
  
public abstract class Document {  
 abstract void addHeader();  
 abstract void addBody();  
 abstract void addFooter();  
 abstract void createDocument();  
}

Листинг 39 – код для пр. работы №8 класс HTMLDocument

package task8.TemplateMethod;  
  
public class HTMLDocument extends Document{  
 @Override  
 public void addHeader(){  
 System.*out*.println("Now HTML document has a header tag.");  
 }  
 @Override  
 public void addBody(){  
 System.*out*.println("Now HTML document has a body tag.");  
 }  
 @Override  
 public void addFooter(){  
 System.*out*.println("Now HTML document has a footer tag.");  
 }  
 @Override  
 public void createDocument(){  
 addHeader();  
 addBody();  
 addFooter();  
 }  
}

1. **Результат работы программы**

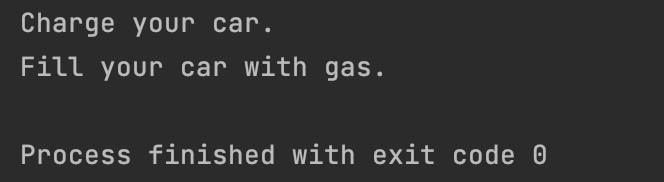
****

Рисунок 12 – Результат работы программы

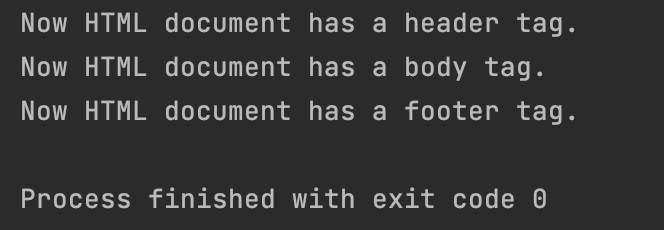
****

Рисунок 13 – Результат работы программы

**Практическая работа №9**

1. **Постановка задачи и персональный вариант**

Цель работы: Знакомство с системой сборки приложения. Gradle.

Постановка задачи: 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.

1. **Код и пояснение к коду**

Листинг 40 – код для пр. работы №9

plugins **{** id 'java'  
**}**group 'org.example'  
version '1.0-SNAPSHOT'  
  
repositories **{** mavenCentral()  
**}**dependencies **{** testImplementation 'org.junit.jupiter:junit-jupiter-api:5.8.1'  
 testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.8.1'  
**}**java **{** sourceCompatibility = JavaVersion.*toVersion*("18")  
 targetCompatibility = JavaVersion.*toVersion*("18")  
**}**test **{** useJUnitPlatform()  
**}**jar **{** baseName("task9")  
 manifest **{** attributes("Main-Class": "org.example.Main")  
 **}  
}**task copyJar(type: Copy)**{** dependsOn(jar)  
 from('build/libs')  
 into('dockerfile')  
**}**task buildDocker(type: Exec)**{** dependsOn(copyJar)  
 commandLine 'docker', 'build', '-t', 'task9', './dockerfile'  
**}**task runDocker(type: Exec)**{** dependsOn(buildDocker)  
 commandLine 'docker', 'run', '-p', '8080:8080', 'task9'  
**}**

1. **Результат работы программы**

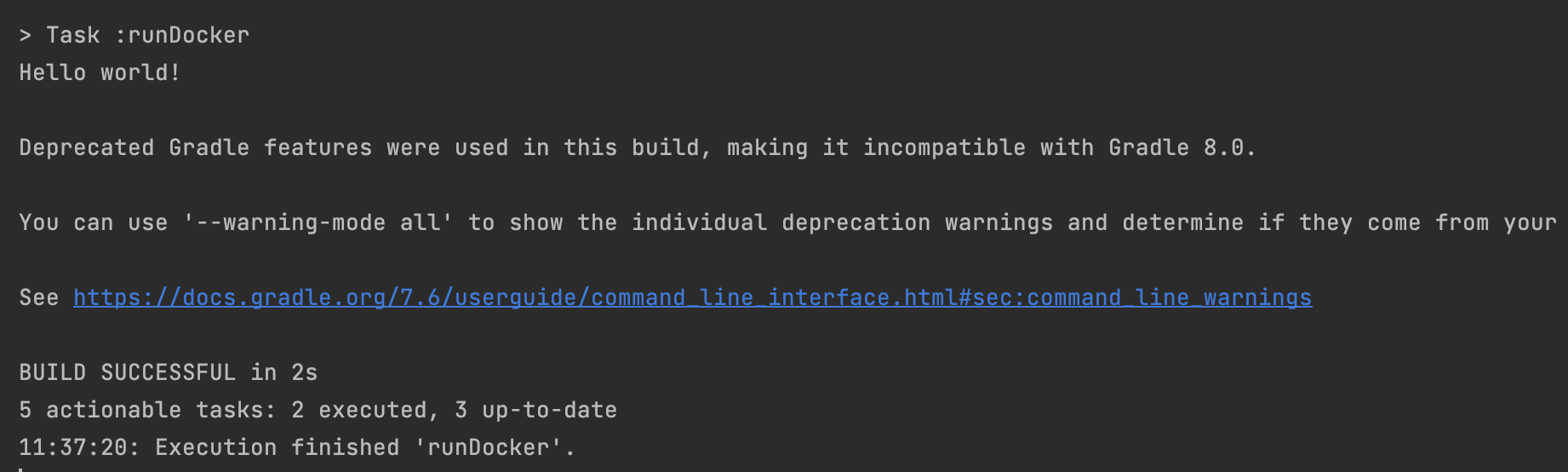


Рисунок 14 – Результат работы программы

**Практическая работа №10**

1. **Постановка задачи и персональный вариант**

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

Постановка задачи: 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. Интерфейс Politician с методом doPolitic(), его имплементации: Trump,Biden, Merkel.

1. **Код и пояснения к коду**

Листинг 41 – код для пр. работы №10

package com.example.demo;  
  
import com.example.config.AppConfig;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.ConfigurableApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
  
import java.util.Scanner;  
  
@SpringBootApplication  
public class DemoApplication {  
  
 public static void main(String[] args) {  
  
 Politician politician;  
 ApplicationContext applicationContext = new AnnotationConfigApplicationContext(AppConfig.class);  
 Scanner scanner = new Scanner(System.*in*);  
 String name = scanner.nextLine();  
 if (name.equals("Trump")){  
 politician = applicationContext.getBean(Trump.class);  
 } else if (name.equals("biden")) {  
 politician = applicationContext.getBean(Biden.class);  
 } else{  
 politician = applicationContext.getBean(Merkel.class);  
 }  
 politician.doPolitic();  
 }  
}

Листинг 42 – код для пр. работы №10 интерфейс Politician

package com.example.demo;  
  
public interface Politician {  
 void doPolitic();  
}

Листинг 43 – код для пр. работы №10 класс Biden

package com.example.demo;  
  
public class Biden implements Politician {  
 @Override  
 public void doPolitic() {  
 System.*out*.println("SODA!!!!");  
 }  
}

Листинг 44 – код для пр. работы №10 класс Merkel

package com.example.demo;  
  
public class Merkel implements Politician{  
 @Override  
 public void doPolitic() {  
 System.*out*.println("Bla-bla-bla.");  
 }  
}

Листинг 45 – код для пр. работы №10 класс Trump

package com.example.demo;  
  
public class Trump implements Politician {  
 @Override  
 public void doPolitic() {  
 System.*out*.println("Obamna");  
 }  
}

Листинг 46 – код для пр. работы №10 класс AppConfig

package com.example.config;  
  
import com.example.demo.\*;  
import com.example.demo.task12.FileManipulator;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class AppConfig {  
  
 @Bean  
 public Politician Merkel(){  
 return new Merkel();  
 }  
  
 @Bean  
 public Politician Biden(){  
 return new Biden();  
 }  
 @Bean  
 public Politician Trump(){  
 return new Trump();  
 }  
 @Bean  
 public Student student(){  
 return new Student();  
 }  
  
}

1. **Результат работы программы**

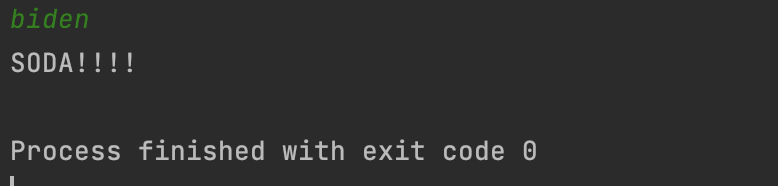


Рисунок 15 – Результат работы программы

**Практическая работа №11**

1. **Постановка задачи и персональный вариант**

Цель работы: Разобраться с использованием Spring boot.

Постановка задачи: 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.

1. **Код и пояснение к коду**

Листинг 47 – код для пр. работы №11 файл pom.xml

<?xml version="1.0" encoding="UTF-8"?>  
dependencies **{** implementation group: 'org.springframework.boot', name: 'spring-boot-starter-actuator', version: '3.0.5'  
 implementation group: 'org.springframework.boot', name: 'spring-boot-starter-validation', version: '3.0.5'  
 implementation group: 'org.springframework.boot', name: 'spring-boot-starter-web', version: '3.0.5'  
 implementation 'org.springframework.boot:spring-boot-starter'  
 implementation 'org.projectlombok:lombok:1.18.20'  
 testImplementation group: 'org.springframework.boot', name: 'spring-boot-starter-test', version: '3.0.5'  
 implementation group: 'org.springframework.boot', name: 'spring-boot-starter-thymeleaf', version: '3.0.1'  
 implementation group: 'org.springframework.boot', name: 'spring-boot-starter-data-jpa', version: '3.0.4'  
 implementation group: 'org.postgresql', name: 'postgresql', version: '42.5.4'  
**}**

1. **Результат работы программы**



Рисунок 16 – Результат работы программы

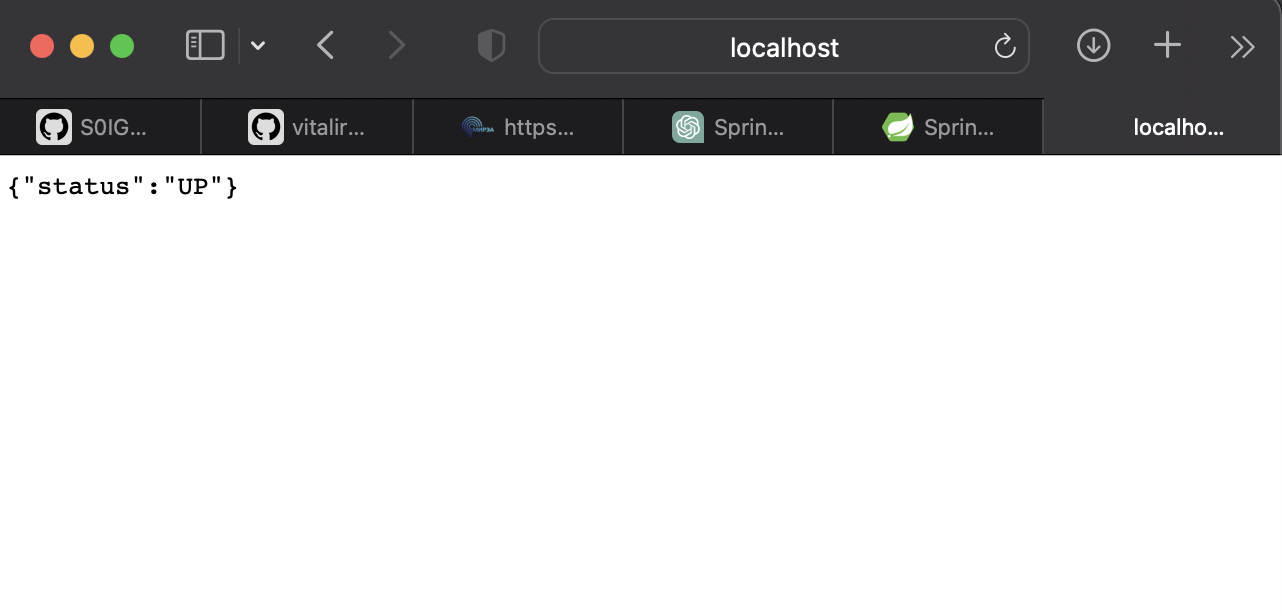


Рисунок 17 – Результат работы программы

**Практическая работа №12**

1. **Постановка задачи и персональный вариант**

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

Постановка задачи: 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.

1. **Код и пояснение к коду**

Листинг 48 – код для пр. работы №12

package com.example.demo.task12;  
  
import jakarta.annotation.PostConstruct;  
import jakarta.annotation.PreDestroy;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
import java.io.File;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.nio.file.Files;  
import java.security.MessageDigest;  
import java.security.NoSuchAlgorithmException;  
  
@SpringBootApplication  
public class FileManipulator {  
  
 private String inputFile = "/Users/tweazy/Desktop/input.txt";  
 private String outputFile = "/Users/tweazy/Desktop/output.txt";  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(FileManipulator.class);  
  
 }  
  
 @PostConstruct  
 public void start() throws IOException, NoSuchAlgorithmException {  
 File file = new File(inputFile);  
  
 if (!file.exists()){  
 createEmptyFile("output.txt");  
 return;  
 }  
  
 String hash = getHash(file);  
  
 writeHashToFile(outputFile, hash);  
 Files.*delete*(file.toPath());  
  
 }  
  
 public void createEmptyFile(String fileName) throws IOException {  
 try(FileWriter writer = new FileWriter(fileName)) {  
 writer.write("null");  
 }  
 }  
  
 public String getHash(File file) throws NoSuchAlgorithmException, IOException {  
 MessageDigest messageDigest = MessageDigest.*getInstance*("SHA-256");  
 byte[] hashBytes = messageDigest.digest(Files.*readAllBytes*(file.toPath()));  
  
 StringBuilder hash = new StringBuilder();  
  
 for (byte hashByte : hashBytes){  
 hash.append(Integer.*toHexString*(0xff & hashByte));  
 }  
 return hash.toString();  
 }  
  
 public void writeHashToFile(String fileName, String hash) throws IOException {  
 try(FileWriter writer = new FileWriter(fileName)){  
 writer.write(hash);  
 }  
 }  
  
  
 @PreDestroy  
 public void end() throws IOException {  
 File file = new File(inputFile);  
  
 Files.*delete*(file.toPath());  
 }  
}

1. **Результат работы программы**



Рисунок 18 – Результат работы программы

****

Рисунок 19 – Результат работы программы

**Практическая работа №13**

1. **Постановка задачи и персональный вариант**

Цель работы: Конфигурирование приложения. Environment.

Постановка задачи: 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.

1. **Код и пояснение к коду**

Листинг 49 – код для пр. работы №13

package com.example.demo;  
  
import com.example.config.AppConfig;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.ConfigurableApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
  
import java.util.Scanner;  
  
@SpringBootApplication  
public class DemoApplication {  
  
 public static void main(String[] args) {  
 ConfigurableApplicationContext context = SpringApplication.*run*(DemoApplication.class, args);  
  
 Student student = context.getBean(Student.class);  
  
 System.*out*.println(student);  
 }  
}

Листинг 50 – код для пр. работы №13

package com.example.demo;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.context.EnvironmentAware;  
import org.springframework.core.env.Environment;  
import org.springframework.stereotype.Component;  
  
@Component  
public class Student implements EnvironmentAware {  
 private Environment environment;  
  
 @Value("${student.name}")  
 private String name;  
  
 @Value("${student.last\_name}")  
 private String last\_name;  
  
 @Value("${student.group}")  
 private String group;  
  
 @Override  
 public String toString() {  
 return "Student{" +  
 "name='" + name + '\'' +  
 ", last\_name='" + last\_name + '\'' +  
 ", group='" + group + '\'' +  
 '}';  
 }  
  
 @Override  
 public void setEnvironment(Environment environment) {  
 this.environment = environment;  
 }  
}

**Практическая работа №14**

1. **Постановка задачи и персональный вариант**

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

Постановка задачи: 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. Создать класс Item с полями name, creationDate, price. Создать класс Order с полями orderDate. Создать классы-контроллеры для создания, удаления объектов и получения всех объектов каждого типа. Сами объекты хранить в памяти.

1. **Код и пояснение к коду**

Листинг 52 – код для пр. работы №14

package com.example.demo.task14.controllers;  
  
import com.example.demo.task14.Index;  
import com.example.demo.task14.Item;  
import org.springframework.web.bind.annotation.\*;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
  
@RestController  
public class ItemController {  
 private final List<Item> Items = new ArrayList<>(){{  
 add(new Item("Item1", 1000, LocalDate.*of*(2023, 4, 13)));  
 add(new Item("Item2", 2000, LocalDate.*of*(2023, 12, 5)));  
 add(new Item("Item3", 3000, LocalDate.*of*(2023, 10, 1)));  
 add(new Item("Item4", 4000, LocalDate.*of*(2023, 5, 15)));  
 add(new Item("Item5", 5000, LocalDate.*of*(2023, 9, 30)));  
 }};  
  
 @GetMapping("/items")  
 public List<Item> getItems(){  
 return Items;  
 }  
 @DeleteMapping("/items")  
 public List<Item> deleteItem(@RequestBody Index index){  
 Items.remove(index.index);  
 return Items;  
 }  
 @PostMapping("/items")  
 public void addItem(@RequestBody Item item){  
 Items.add(item);  
 }  
}

Листинг 53 – код для пр. работы №14

package com.example.demo.task14;  
  
import java.time.LocalDate;  
  
public class Item {  
 private String name;  
 private int price;  
 private LocalDate creationDate;  
  
 public Item(String name, int price, LocalDate creationDate){  
 this.name = name;  
 this.price = price;  
 this.creationDate = creationDate;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getPrice() {  
 return price;  
 }  
  
 public void setPrice(int price) {  
 this.price = price;  
 }  
  
 public LocalDate getCreationDate() {  
 return creationDate;  
 }  
  
 public void setCreationDate(LocalDate creationDate) {  
 this.creationDate = creationDate;  
 }  
}

Листинг 54 – код для пр. работы №14

package com.example.demo.task14.controllers;  
  
  
import com.example.demo.task14.Index;  
import com.example.demo.task14.Order;  
import org.springframework.web.bind.annotation.\*;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
  
@RestController  
public class OrderController {  
  
 private final List<Order> Orders = new ArrayList<>(){{  
 add(new Order(LocalDate.*of*(2023, 9, 13)));  
 add(new Order(LocalDate.*of*(2020, 10, 4)));  
 add(new Order(LocalDate.*of*(2022, 5, 19)));  
 add(new Order(LocalDate.*of*(2021, 4, 30)));  
 add(new Order(LocalDate.*of*(2022, 12, 30)));  
 }};  
  
  
 @GetMapping("/orders")  
 public List<Order> getOrders(){  
 return Orders;  
 }  
 @DeleteMapping("/order")  
 public List<Order> deleteOrder(@RequestBody Index index){  
 Orders.remove(index.index);  
 return Orders;  
 }  
 @PostMapping("/orders")  
 public void addOrder(@RequestBody Order order){  
 Orders.add(order);  
 }  
}

Листинг 55 – код для пр. работы №14

package com.example.demo.task14;  
  
  
import java.time.LocalDate;  
  
public class Order {  
 private LocalDate orderDate;  
  
 public Order(LocalDate orderDate) {  
 this.orderDate = orderDate;  
 }  
  
 public LocalDate getOrderDate() {  
 return orderDate;  
 }  
  
 public void setOrderDate(LocalDate orderDate) {  
 this.orderDate = orderDate;  
 }  
}

Листинг 56 – код для пр. работы №14

package com.company.practics.practic\_14;  
  
import org.springframework.core.io.ClassPathResource;  
import org.springframework.core.io.Resource;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.ResponseBody;  
  
import java.io.IOException;  
package com.example.demo.task14.controllers;  
  
  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.GetMapping;  
  
import java.io.IOException;  
  
@Controller  
public class HomeController {  
  
 @GetMapping("/home")  
 public String getHomePage() throws IOException{  
 return "home";  
 }  
}

1. **Результат работы программы**

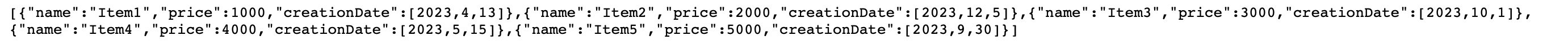
****

Рисунок 21 – Результат работы программы

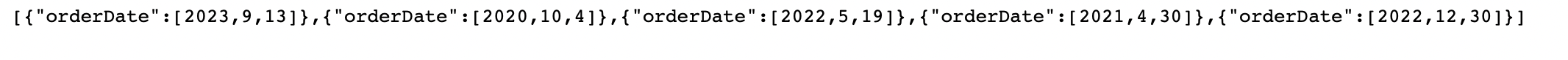


Рисунок 23 – Результат работы программы

**Практическая работа №15**

1. **Постановка задачи и персональный вариант**

Цель работы: Использование Hibernate в Spring Framework.

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

1. **Код и пояснение к коду**

Листинг 57 – код для пр. работы №15

package com.example.demo.task15.controllers;  
  
import com.example.demo.task15.Item;  
import com.example.demo.task15.repository.ItemRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
  
public class ItemController {  
 private final ItemRepository itemRepository;  
  
 @Autowired  
 public ItemController(ItemRepository itemRepository) {  
 this.itemRepository = itemRepository;  
 }  
  
  
 @GetMapping("/items")  
 public List<Item> getItems(){  
 return itemRepository.findAll();  
 }  
 @DeleteMapping("/items")  
 public List<Item> deleteItem(@RequestBody Item item){  
 itemRepository.delete(item);  
 return itemRepository.findAll();  
 }  
 @PostMapping("/items")  
 public void addItem(@RequestBody Item item){  
 itemRepository.save(item);  
 }  
}

Листинг 58 – код для пр. работы №15

package com.example.demo.task15;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.Id;  
  
import java.time.LocalDate;  
  
@Entity  
public class Item {  
 @Id  
 private Integer id;  
 private String name;  
 private Integer price;  
 private LocalDate creationDate = LocalDate.*now*();  
  
 public Item(String name, Integer price) {  
 this.name = name;  
 this.price = price;  
 }  
 public Item(){}  
  
 public Integer getPrice() {  
 return price;  
 }  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public void setPrice(Integer price) {  
 this.price = price;  
 }  
  
 public void setCreationDate(LocalDate creationDate) {  
 this.creationDate = creationDate;  
 }  
  
 public Integer getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public LocalDate getCreationDate() {  
 return creationDate;  
 }  
}

Листинг 59 – код для пр. работы №15

package com.example.demo.task15.controllers;  
  
  
import com.example.demo.task15.Order;  
import com.example.demo.task15.repository.OrderRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
public class OrderController {  
 private final OrderRepository orderRepository;  
  
 @Autowired  
 public OrderController(OrderRepository orderRepository) {  
 this.orderRepository = orderRepository;  
 }  
  
 @GetMapping("/orders")  
 public List<Order> getOrders(){  
 return orderRepository.findAll();  
 }  
 @DeleteMapping("/orders")  
 public List<Order> deleteOrder(@RequestBody Order order){  
 orderRepository.delete(order);  
 return orderRepository.findAll();  
 }  
 @PostMapping("/orders")  
 public void addUser(@RequestBody Order order){  
 orderRepository.save(order);  
 }  
}

Листинг 60 – код для пр. работы №15

package com.example.demo.task15;  
  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.Id;  
import jakarta.persistence.Table;  
  
import java.time.LocalDate;  
  
@Entity  
@Table(name = "\_order")  
public class Order {  
 @Id  
 private Integer id;  
 private LocalDate orderDate = LocalDate.*now*();  
  
 public Order(Integer id) {  
 this.id = id;  
 }  
  
 public Order() {  
 }  
  
 public Integer getId() {  
 return id;  
 }  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
  
 public LocalDate getOrderDate() {  
 return orderDate;  
 }  
  
 public void setOrderDate(LocalDate orderDate) {  
 this.orderDate = orderDate;  
 }  
}

1. **Результат работы программы**

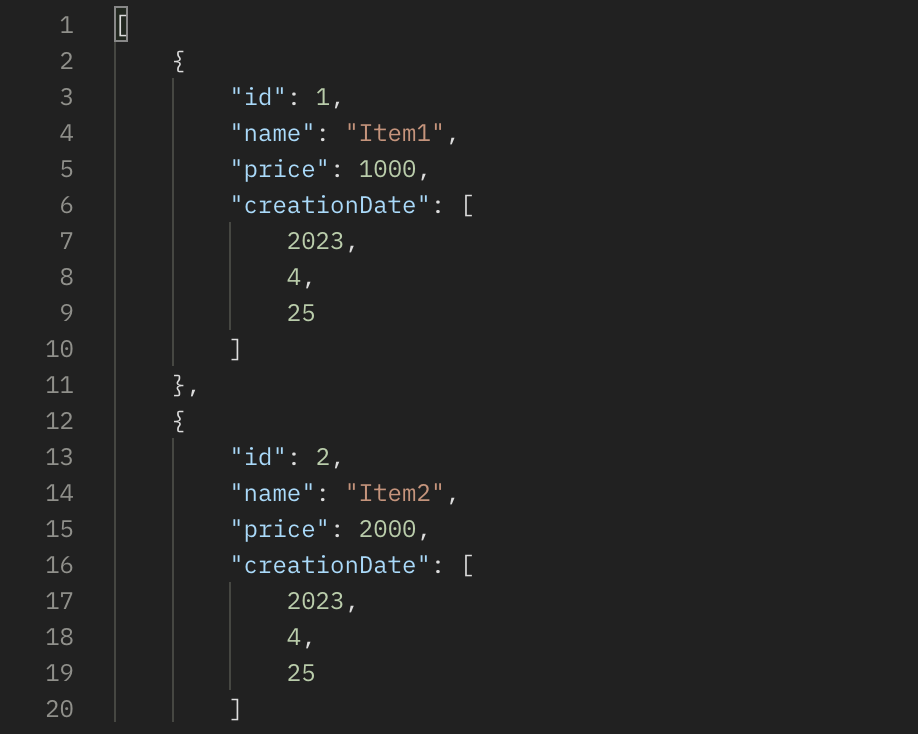
****

Рисунок 24 – Результат работы программы

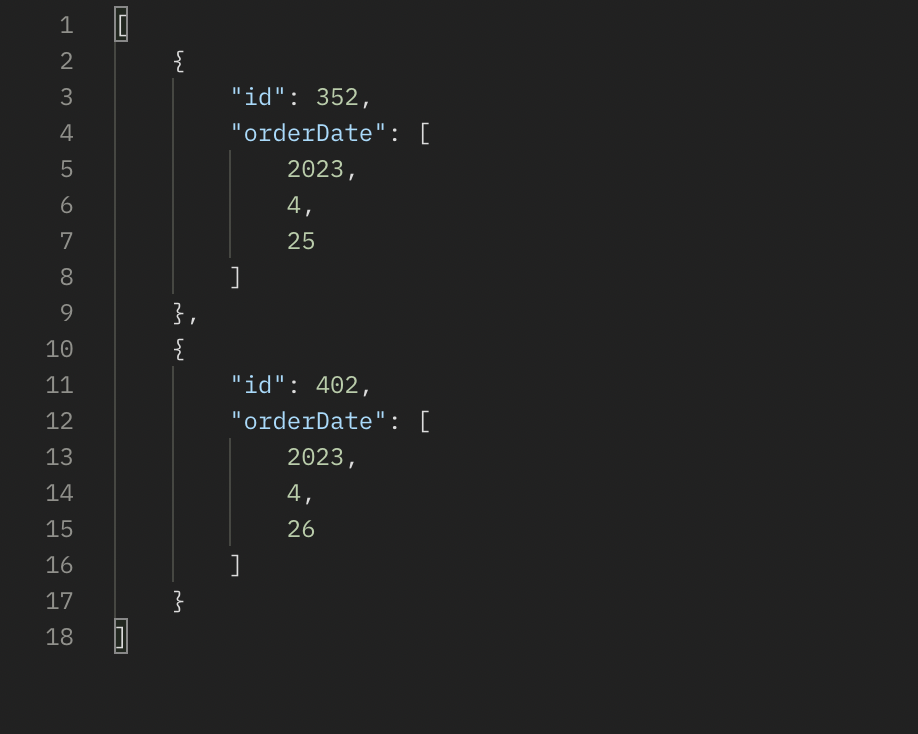


Рисунок 25 – Результат работы программы

**Практическая работа №16**

1. **Постановка задачи и персональный вариант**

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

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

1. **Код и пояснение к коду**

Листинг 61 – код для пр. работы №16

package com.example.demo.task16;  
  
import com.fasterxml.jackson.annotation.JsonIgnore;  
import jakarta.persistence.\*;  
  
import java.time.LocalDate;  
  
@Entity  
  
public class Item {  
 @Id  
 private Integer id;  
 private String name;  
 private Integer price;  
 private LocalDate creationDate = LocalDate.*now*();  
  
 @ManyToOne  
 @JsonIgnore  
 @JoinColumn(name = "order\_id")  
 public Order order;  
  
 public Item(String name, Integer price) {  
 this.name = name;  
 this.price = price;  
 }  
 public Item(){}  
  
 public Integer getPrice() {  
 return price;  
 }  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public void setPrice(Integer price) {  
 this.price = price;  
 }  
  
 public void setCreationDate(LocalDate creationDate) {  
 this.creationDate = creationDate;  
 }  
  
 public Integer getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public LocalDate getCreationDate() {  
 return creationDate;  
 }  
  
 public Order getOrder() {  
 return order;  
 }  
  
 public void setOrder(Order order) {  
 this.order = order;  
 }  
}

Листинг 62 – код для пр. работы №16

package com.example.demo.task16;  
  
import jakarta.persistence.\*;  
  
import java.time.LocalDate;  
import java.util.List;  
  
@Entity  
@Table(name = "\_order")  
public class Order {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Integer id;  
  
 private LocalDate orderDate = LocalDate.*now*();  
  
 @OneToMany(mappedBy = "order")  
 public List<Item> items;  
  
 public Order(Integer id) {  
 this.id = id;  
 }  
  
 public Order() {  
 }  
  
 public Integer getId() {  
 return id;  
 }  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
  
 public LocalDate getOrderDate() {  
 return orderDate;  
 }  
  
 public void setOrderDate(LocalDate orderDate) {  
 this.orderDate = orderDate;  
 }  
  
 public List<Item> getItems() {  
 return items;  
 }  
  
 public void setItems(List<Item> items) {  
 this.items = items;  
 }  
}

1. **Результат работы программы**

****

Рисунок 26 – Результат работы программы

**Практическая работа №17**

1. **Постановка задачи и персональный вариант**

Цель работы: Знакомство с Criteria API в Hibernate.

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

1. **Код и пояснение к коду**

Листинг 63 – код для пр. работы №17

package com.example.demo.task17.controllers;  
  
  
import com.example.demo.task17.Item;  
import com.example.demo.task17.Order;  
import com.example.demo.task17.repository.ItemRepository;  
import com.example.demo.task17.repository.OrderRepository;  
import jakarta.persistence.EntityManager;  
import jakarta.persistence.TypedQuery;  
import jakarta.persistence.criteria.CriteriaBuilder;  
import jakarta.persistence.criteria.CriteriaQuery;  
import jakarta.persistence.criteria.Predicate;  
import jakarta.persistence.criteria.Root;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
@RestController  
public class OrderController {  
 private final OrderRepository orderRepository;  
 private final ItemRepository itemRepository;  
 private final EntityManager entityManager;  
  
  
 @Autowired  
 public OrderController(OrderRepository orderRepository, ItemRepository itemRepository, EntityManager entityManager) {  
 this.orderRepository = orderRepository;  
 this.itemRepository = itemRepository;  
 this.entityManager = entityManager;  
 }  
  
 public CriteriaBuilder getCriteriaBuilder() {  
 return entityManager.getCriteriaBuilder();  
 }  
  
 public TypedQuery<Order> createQuery(CriteriaQuery<Order> criteriaQuery) {  
 return entityManager.createQuery(criteriaQuery);  
 }  
  
 @GetMapping("/orders")  
 public List<Order> getOrders(@RequestParam(required = false) LocalDate from,  
 @RequestParam(required = false) LocalDate to) {  
  
 CriteriaBuilder cb = getCriteriaBuilder();  
 CriteriaQuery<Order> query = cb.createQuery(Order.class);  
 Root<Order> root = query.from(Order.class);  
  
 List<Predicate> predicates = new ArrayList<>();  
  
 if (from != null) {  
 predicates.add(cb.greaterThanOrEqualTo(root.get("orderDate"), from));  
 }  
 if (to != null) {  
 predicates.add(cb.lessThanOrEqualTo(root.get("orderDate"), to));  
 }  
  
 if (!predicates.isEmpty()) {  
 query.where(predicates.toArray(new Predicate[0]));  
 }  
  
 return createQuery(query).getResultList();  
 }  
  
 @DeleteMapping("/orders")  
 public List<Order> deleteOrder(@RequestBody Order order){  
 orderRepository.delete(order);  
 return orderRepository.findAll();  
 }  
  
 @GetMapping("/order/{orderId}")  
 public @ResponseBody Optional<Order> getOrder(@PathVariable("orderId") Integer orderId){  
 return orderRepository.findById(orderId);  
 }  
 @PostMapping("/orders")  
 public void addOrder(@RequestBody Order order){  
 System.*out*.println(order.items);  
 order = orderRepository.save(order);  
 for ( Item item : order.items){  
 item.setOrder(order);  
 }  
 order.setItems(itemRepository.saveAll(order.items));  
 }  
}

1. **Результат работы программы**

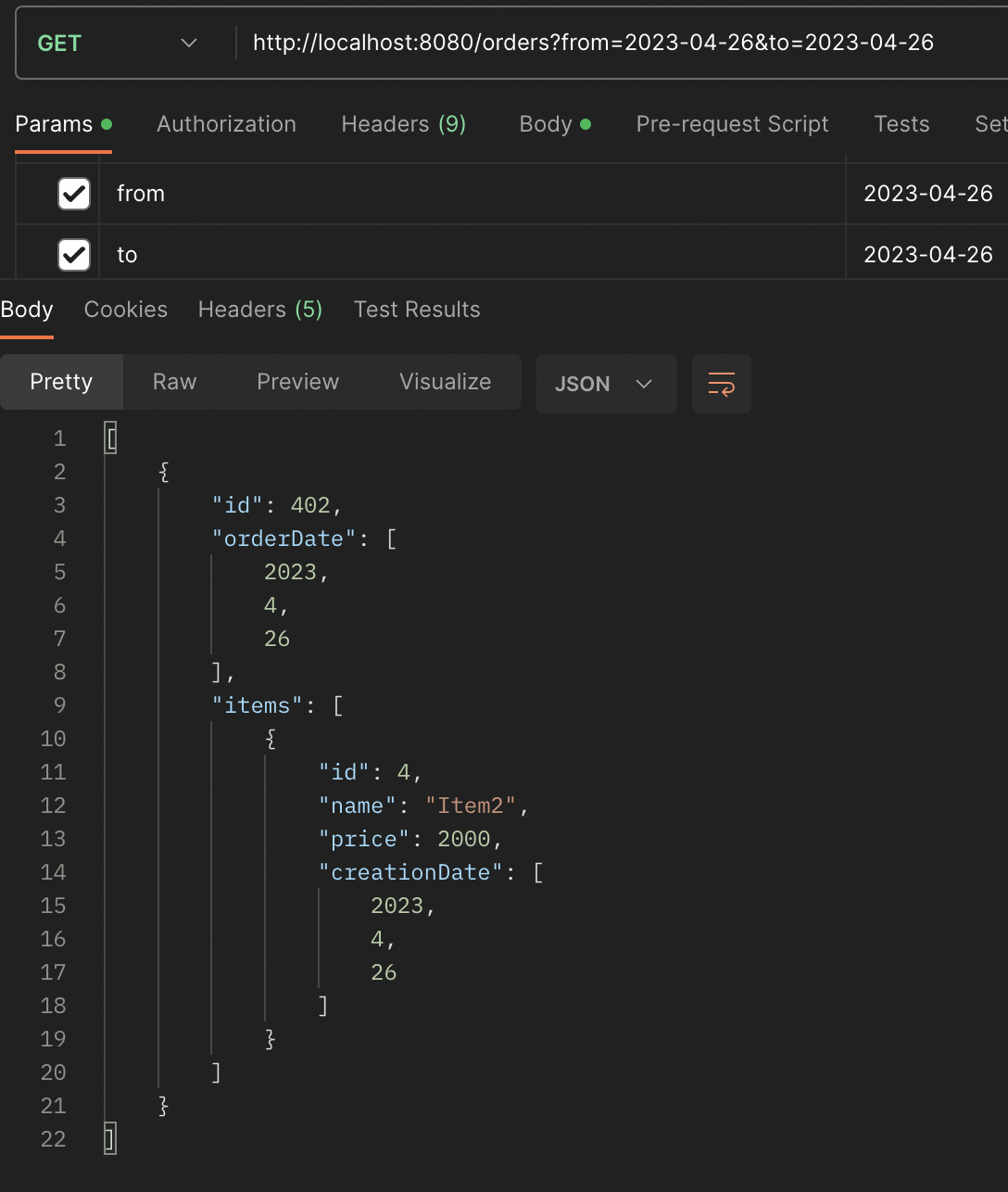
****

Рисунок 27 – Результат работы программы

**Практическая работа №18**

1. **Постановка задачи и персональный вариант**

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

Постановка задачи: 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.

1. **Код и пояснение к коду**

Листинг 64 – код для пр. работы №18

package com.example.demo.task18.services;  
  
import com.example.demo.task18.Item;  
  
import java.time.LocalDate;  
import java.util.List;  
import java.util.Optional;  
  
public interface ItemService {  
 Item saveItem(Item item);  
 List<Item> saveAll(List<Item> items);  
 void deleteItem(Item item);  
 Optional<Item> findById(Integer id);  
 List<Item> findAll();  
 List<Item> findAll(String name, Integer price, LocalDate creationDate);  
}

Листинг 65 – код для пр. работы №18

package com.example.demo.task18.services.impl;  
  
import com.example.demo.task18.repository.ItemRepository;  
import com.example.demo.task18.Item;  
import com.example.demo.task18.services.ItemService;  
import jakarta.persistence.TypedQuery;  
import jakarta.persistence.criteria.\*;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
public class ItemServiceImpl implements ItemService{  
  
 private final ItemRepository itemRepository;  
 private final CriteriaAPI criteriaAPI;  
  
 @Autowired  
 public ItemServiceImpl(ItemRepository itemRepository, CriteriaAPI criteriaAPI) {  
 this.itemRepository = itemRepository;  
 this.criteriaAPI = criteriaAPI;  
 }  
  
 @Override  
 public Item saveItem(Item item) {  
 return itemRepository.save(item);  
 }  
  
 @Override  
 public void deleteItem(Item item) {  
 itemRepository.delete(item);  
 }  
  
 @Override  
 public List<Item> saveAll(List<Item> items) {  
 return itemRepository.saveAll(items);  
 }  
  
 @Override  
 public Optional<Item> findById(Integer id) {  
 return itemRepository.findById(id);  
 }  
  
 @Override  
 public List<Item> findAll() {  
 return itemRepository.findAll();  
 }  
  
 @Override  
 public List<Item> findAll(String name, Integer price, LocalDate creationDate) {  
 CriteriaBuilder criteriaBuilder = criteriaAPI.getCriteriaBuilder();  
 CriteriaQuery<Item> query = criteriaBuilder.createQuery(Item.class);  
 Root<Item> root = query.from(Item.class);  
  
 List<Predicate> predicates = new ArrayList<>();  
  
 if (name != null){  
 predicates.add(criteriaBuilder.like(root.get("name"), name + "%"));  
 }  
 if (price != null){  
 predicates.add(criteriaBuilder.equal(root.get("price"), price));  
 }  
 if (creationDate != null){  
 predicates.add(criteriaBuilder.equal(root.get("creationDate"), creationDate));  
 }  
  
 query.select(root).where(predicates.toArray(new Predicate[]{}));  
 TypedQuery<Item> typedQuery = criteriaAPI.getEntityManager().createQuery(query);  
 return typedQuery.getResultList();  
 }  
  
}

Листинг 66 – код для пр. работы №18

package com.example.demo.task18.services;  
  
import com.example.demo.task18.Order;  
  
import java.time.LocalDate;  
import java.util.List;  
import java.util.Optional;  
  
public interface OrderService {  
 Order saveOrder(Order order);  
 List<Order> saveAll(List<Order> orders);  
 void deleteOrder(Order order);  
 Optional<Order> findById(Integer id);  
 List<Order> findAll();  
 List<Order> findAll(LocalDate to, LocalDate from);  
}

Листинг 67 – код для пр. работы №18

package com.example.demo.task18.services.impl;  
  
import com.example.demo.task18.Item;  
import com.example.demo.task18.repository.OrderRepository;  
import com.example.demo.task18.Order;  
import com.example.demo.task18.services.ItemService;  
import com.example.demo.task18.services.OrderService;  
import jakarta.persistence.TypedQuery;  
import jakarta.persistence.criteria.CriteriaBuilder;  
import jakarta.persistence.criteria.CriteriaQuery;  
import jakarta.persistence.criteria.Predicate;  
import jakarta.persistence.criteria.Root;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
public class OrderServiceImpl implements OrderService {  
 private final OrderRepository orderRepository;  
 private final CriteriaAPI criteriaAPI;  
 private final ItemService itemService;  
  
 @Autowired  
 public OrderServiceImpl(OrderRepository orderRepository, CriteriaAPI criteriaAPI, ItemService itemService) {  
 this.orderRepository = orderRepository;  
 this.criteriaAPI = criteriaAPI;  
 this.itemService = itemService;  
 }  
  
 @Override  
 public Order saveOrder(Order order) {  
 order = orderRepository.save(order);  
 for ( Item item : order.items){  
 item.setOrder(order);  
 }  
 order.setItems(itemService.saveAll(order.items));  
 return orderRepository.save(order);  
 }  
  
 @Override  
 public List<Order> saveAll(List<Order> orders) {  
 return orderRepository.saveAll(orders);  
 }  
  
 @Override  
 public void deleteOrder(Order order) {  
 orderRepository.delete(order);  
 }  
  
 @Override  
 public Optional<Order> findById(Integer id) {  
 return orderRepository.findById(id);  
 }  
  
 @Override  
 public List<Order> findAll() {  
 return orderRepository.findAll();  
 }  
  
 @Override  
 public List<Order> findAll(LocalDate from, LocalDate to) {  
 CriteriaBuilder criteriaBuilder = criteriaAPI.getCriteriaBuilder();  
 CriteriaQuery<Order> query = criteriaBuilder.createQuery(Order.class);  
 Root<Order> root = query.from(Order.class);  
  
 List<Predicate> predicates = new ArrayList<>();  
  
 if (from != null) {  
 predicates.add(criteriaBuilder.greaterThanOrEqualTo(root.get("orderDate"), from));  
 }  
 if (to != null) {  
 predicates.add(criteriaBuilder.lessThanOrEqualTo(root.get("orderDate"), to));  
 }  
  
 if (!predicates.isEmpty()) {  
 query.where(predicates.toArray(new Predicate[0]));  
 }  
  
 TypedQuery<Order> typedQuery = criteriaAPI.getEntityManager().createQuery(query);  
 return typedQuery.getResultList();  
 }  
}

1. **Результат работы программы**

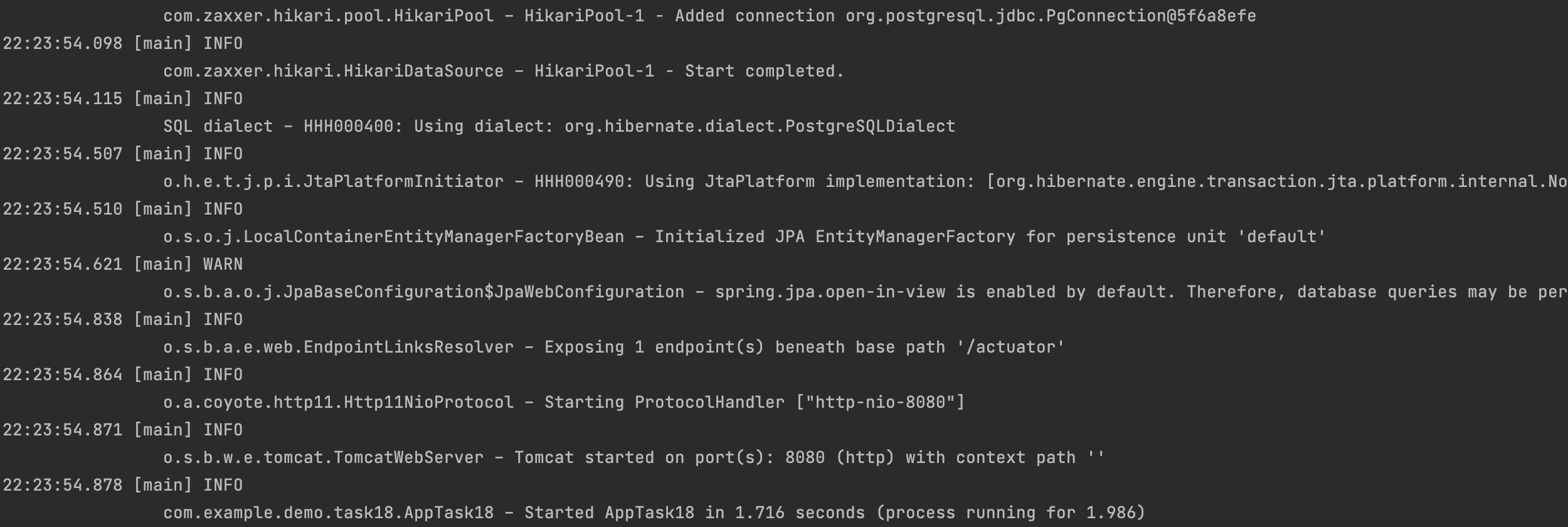
****

Рисунок 28 – Результат работы программы.

**Практическая работа №19**

1. **Постановка задачи и персональный вариант**

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

Постановка задачи: Создать файл logback.xml, добавить логирование во все методы классов- сервисов.

1. **Код и пояснение к коду**

Листинг 68 – код для пр. работы №19

<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>

Листинг 69 – код для пр. работы №19

package com.example.javaframeworktemplates.task19.services.impl;  
  
import com.example.javaframeworktemplates.task19.Item;  
import com.example.javaframeworktemplates.task19.repository.ItemRepository;  
import com.example.javaframeworktemplates.task19.services.ItemService;  
import jakarta.persistence.TypedQuery;  
import jakarta.persistence.criteria.CriteriaBuilder;  
import jakarta.persistence.criteria.CriteriaQuery;  
import jakarta.persistence.criteria.Predicate;  
import jakarta.persistence.criteria.Root;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
@Slf4j  
public class ItemServiceImpl implements ItemService {  
  
 private final ItemRepository itemRepository;  
 private final CriteriaAPI criteriaAPI;  
  
 @Autowired  
 public ItemServiceImpl(ItemRepository itemRepository, CriteriaAPI criteriaAPI) {  
 this.itemRepository = itemRepository;  
 this.criteriaAPI = criteriaAPI;  
 }  
  
 @Override  
 public Item saveItem(Item item) {  
 *log*.info("Saved item: {}", item);  
 return itemRepository.save(item);  
 }  
  
 @Override  
 public void deleteItem(Item item) {  
 *log*.info("Deleted item with id: {}", item.getId());  
 itemRepository.delete(item);  
 }  
  
 @Override  
 public List<Item> saveAll(List<Item> items) {  
 return itemRepository.saveAll(items);  
 }  
  
 @Override  
 public Optional<Item> findById(Integer id) {  
 *log*.info("Find item with id: {}", id);  
 return itemRepository.findById(id);  
 }  
  
 @Override  
 public List<Item> findAll() {  
 *log*.info("Find all items");  
 return itemRepository.findAll();  
 }  
  
 @Override  
 public List<Item> findAll(String name, Integer price, LocalDate creationDate) {  
 *log*.info("Find all items");  
 CriteriaBuilder criteriaBuilder = criteriaAPI.getCriteriaBuilder();  
 CriteriaQuery<Item> query = criteriaBuilder.createQuery(Item.class);  
 Root<Item> root = query.from(Item.class);  
  
 List<Predicate> predicates = new ArrayList<>();  
  
 if (name != null){  
 predicates.add(criteriaBuilder.like(root.get("name"), name + "%"));  
 }  
 if (price != null){  
 predicates.add(criteriaBuilder.equal(root.get("price"), price));  
 }  
 if (creationDate != null){  
 predicates.add(criteriaBuilder.equal(root.get("creationDate"), creationDate));  
 }  
  
 query.select(root).where(predicates.toArray(new Predicate[]{}));  
 TypedQuery<Item> typedQuery = criteriaAPI.getEntityManager().createQuery(query);  
 return typedQuery.getResultList();  
 }  
  
}

Листинг 70 – код для пр. работы №19

package com.example.javaframeworktemplates.task19.services.impl;  
  
import com.example.javaframeworktemplates.task19.Item;  
import com.example.javaframeworktemplates.task19.Order;  
import com.example.javaframeworktemplates.task19.repository.OrderRepository;  
import com.example.javaframeworktemplates.task19.services.ItemService;  
import com.example.javaframeworktemplates.task19.services.OrderService;  
import jakarta.persistence.TypedQuery;  
import jakarta.persistence.criteria.CriteriaBuilder;  
import jakarta.persistence.criteria.CriteriaQuery;  
import jakarta.persistence.criteria.Predicate;  
import jakarta.persistence.criteria.Root;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
@Slf4j  
public class OrderServiceImpl implements OrderService {  
 private final OrderRepository orderRepository;  
 private final CriteriaAPI criteriaAPI;  
 private final ItemService itemService;  
  
 @Autowired  
 public OrderServiceImpl(OrderRepository orderRepository, CriteriaAPI criteriaAPI, ItemService itemService) {  
 this.orderRepository = orderRepository;  
 this.criteriaAPI = criteriaAPI;  
 this.itemService = itemService;  
 }  
  
 @Override  
 public Order saveOrder(Order order) {  
 order = orderRepository.save(order);  
 for ( Item item : order.items){  
 item.setOrder(order);  
 }  
 order.setItems(itemService.saveAll(order.items));  
 *log*.info("Saved order with ID: {} ", order.getId());  
 return orderRepository.save(order);  
 }  
  
 @Override  
 public List<Order> saveAll(List<Order> orders) {  
 return orderRepository.saveAll(orders);  
 }  
  
 @Override  
 public void deleteOrder(Order order) {  
 *log*.info("Deleted order with ID: {}", order.getId());  
 orderRepository.delete(order);  
 }  
  
 @Override  
 public Optional<Order> findById(Integer id) {  
 *log*.info("Find order with ID: {}", id);  
 return orderRepository.findById(id);  
 }  
  
 @Override  
 public List<Order> findAll() {  
 *log*.info("Find all orders");  
 return orderRepository.findAll();  
 }  
  
 @Override  
 public List<Order> findAll(LocalDate from, LocalDate to) {  
 *log*.info("Find all orders");  
 CriteriaBuilder criteriaBuilder = criteriaAPI.getCriteriaBuilder();  
 CriteriaQuery<Order> query = criteriaBuilder.createQuery(Order.class);  
 Root<Order> root = query.from(Order.class);  
  
 List<Predicate> predicates = new ArrayList<>();  
  
 if (from != null) {  
 predicates.add(criteriaBuilder.greaterThanOrEqualTo(root.get("orderDate"), from));  
 }  
 if (to != null) {  
 predicates.add(criteriaBuilder.lessThanOrEqualTo(root.get("orderDate"), to));  
 }  
  
 if (!predicates.isEmpty()) {  
 query.where(predicates.toArray(new Predicate[0]));  
 }  
  
 TypedQuery<Order> typedQuery = criteriaAPI.getEntityManager().createQuery(query);  
 return typedQuery.getResultList();  
 }  
}

1. **Результат работы программы**

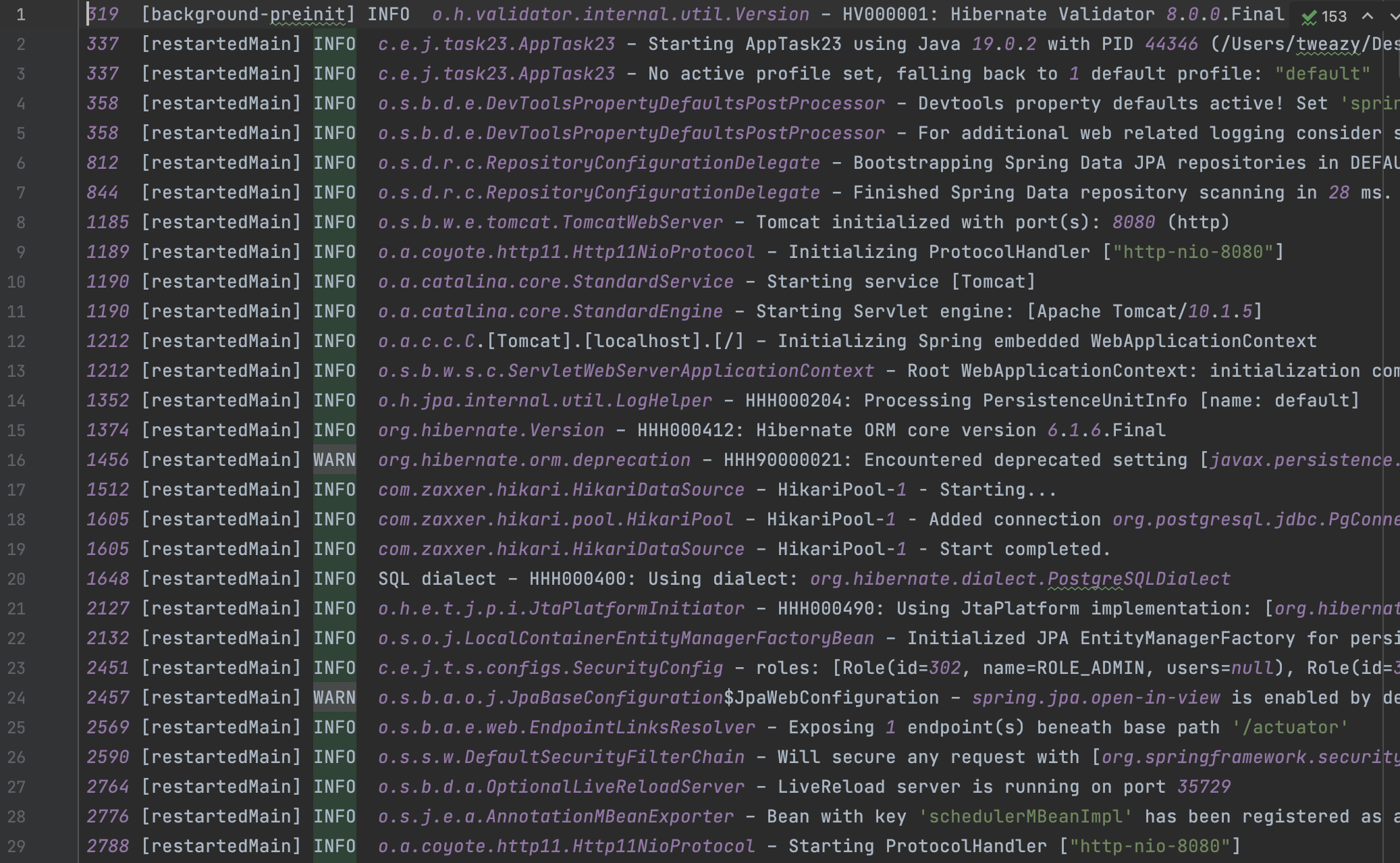
****

Рисунок 29 – Результат работы программы

**Практическая работа №20**

1. **Постановка задачи и персональный вариант**

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

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

1. **Код и пояснение к коду**

Листинг 71 – код для пр. работы №20 добавлен класс aspect

package com.example.javaframeworktemplates.task20;  
  
  
import lombok.extern.slf4j.Slf4j;  
import org.aspectj.lang.JoinPoint;  
import org.aspectj.lang.annotation.After;  
import org.aspectj.lang.annotation.Before;  
import org.aspectj.lang.annotation.Pointcut;  
import org.springframework.stereotype.Component;  
  
@Slf4j  
@Component  
@org.aspectj.lang.annotation.Aspect  
public class Aspect {  
 private long startTime;  
  
 @Before("allServiceMethods()()")  
 public void logMethod(JoinPoint joinPoint) {  
 startTime = System.*nanoTime*();  
 *log*.info("The method is now executed: " + joinPoint.getSignature());  
 }  
 @After("allServiceMethods()()")  
 public void endMethod(JoinPoint joinPoint) {  
 *log*.info("The method "+ joinPoint.getSignature() +" ended in:" +  
 ((System.*nanoTime*() - startTime) / 1\_000\_000\_000) +" ms");  
  
 }  
 @Pointcut("within(com.example.javaframeworktemplates.task20.services.impl.\*)")  
 public void allServiceMethods(){}  
}

1. **Результат работы программы**

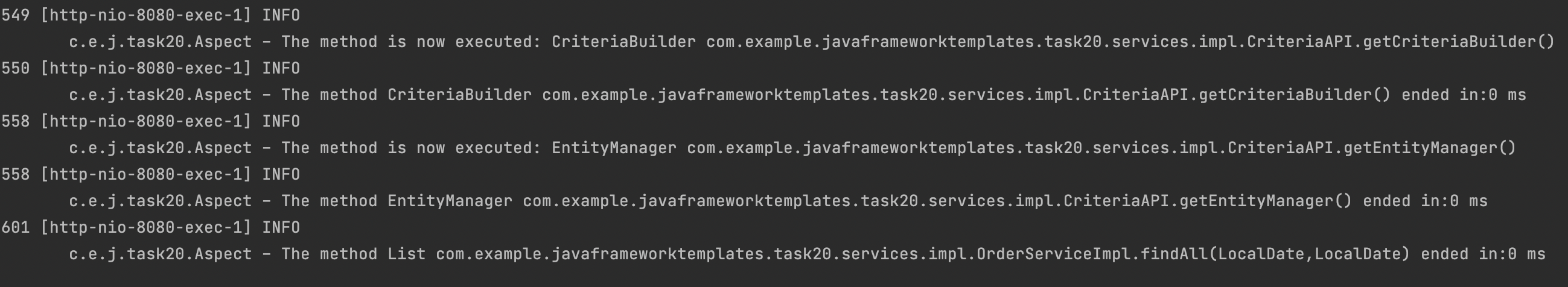
****

Рисунок 30 – Результат работы программы

**Практическая работа №21**

1. **Постановка задачи и персональный вариант**

Цель работы: Проксирование. Аннотация Transactional. Аннотация Async.

Постановка задачи: 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.

1. **Код и пояснение к коду**

Листинг 72 – код для пр. работы №21

package com.example.javaframeworktemplates.task21.services;  
  
public interface EmailService {  
 void sendEmail(String emailTo, String subject, String message);  
}

Листинг 73 – код для пр. работы №21

package com.example.javaframeworktemplates.task21.services.impl;  
  
import com.example.javaframeworktemplates.task21.services.EmailService;  
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 EmailServiceImpl implements EmailService {  
  
 @Value("${spring.mail.username}")  
 private String username;  
 private final JavaMailSender mailSender;  
  
 @Autowired  
 public EmailServiceImpl(JavaMailSender mailSender){  
 this.mailSender = mailSender;  
 }  
  
 @Async  
 @Override  
 public void sendEmail(String emailTo, String subject, String message) {  
 SimpleMailMessage mailMessage = new SimpleMailMessage();  
 mailMessage.setFrom(username);  
 mailMessage.setTo(emailTo);  
 mailMessage.setSubject(subject);  
 mailMessage.setText(message);  
  
 mailSender.send(mailMessage);  
 }  
}

Листинг 74 – код для пр. работы №21

spring.mail.host=smtp.mail.ru  
spring.mail.username=zhavoronkov03@inbox.ru  
spring.mail.password=  
spring.mail.port=465  
spring.mail.protocol=smtps

1. **Результат работы программы**

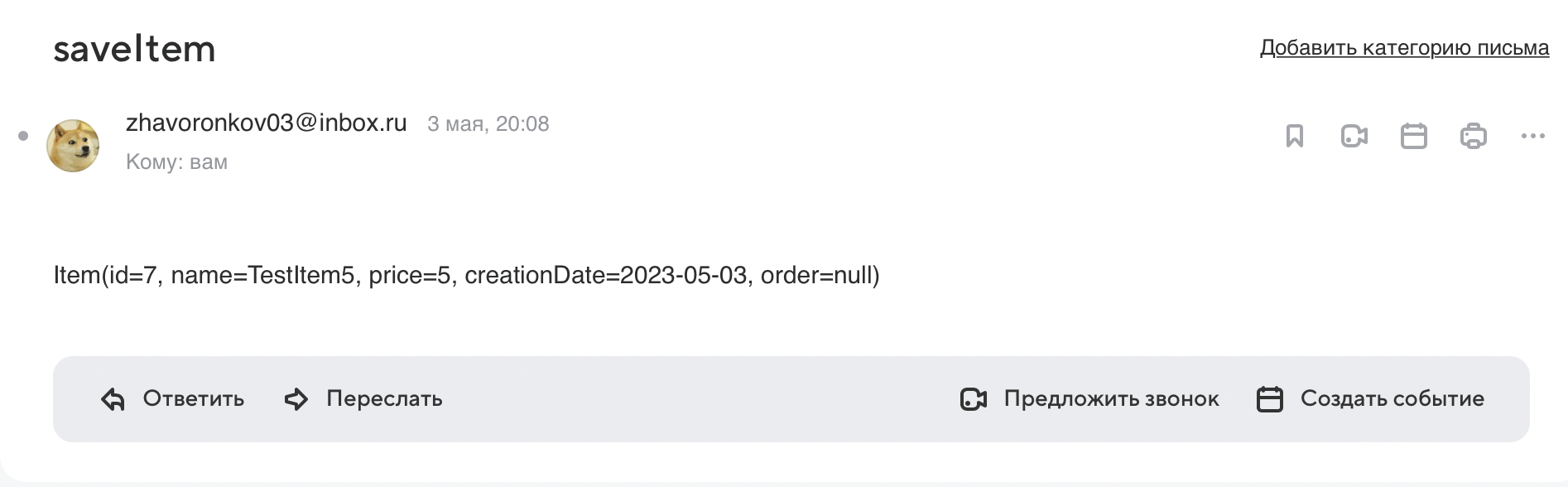
****

Рисунок 31 – Результат работы программы

**Практическая работа №22**

1. **Постановка задачи и персональный вариант**

Цель работы: Планирование заданий. Scheduler в Spring.

Постановка задачи: 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).

1. **Код и пояснение к коду**

Листинг 75 – код для пр. работы №22

package com.example.javaframeworktemplates.task22.services;  
  
import java.io.IOException;  
  
public interface SchedulerService {  
 void cleanDirectory() throws IOException;  
}

Листинг 76 – код для пр. работы №22

package com.example.javaframeworktemplates.task22.services.impl;  
  
import com.example.javaframeworktemplates.task22.entities.Item;  
import com.example.javaframeworktemplates.task22.entities.Order;  
import com.example.javaframeworktemplates.task22.services.ItemService;  
import com.example.javaframeworktemplates.task22.services.OrderService;  
import com.example.javaframeworktemplates.task22.services.SchedulerService;  
import lombok.extern.slf4j.Slf4j;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.scheduling.annotation.Scheduled;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.io.FileWriter;  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.util.List;  
  
@Service  
@Slf4j  
public class SchedulerServiceImpl implements SchedulerService {  
 private final OrderService orderService;  
 private final ItemService itemService;  
  
  
 private final Path orderSrc = Path.*of*("/Users/tweazy/Desktop/order.txt");  
 private final Path itemSrc = Path.*of*("/Users/tweazy/Desktop/item.txt");  
  
 @Autowired  
 public SchedulerServiceImpl(OrderService orderService, ItemService itemService){  
 this.itemService = itemService;  
 this.orderService = orderService;  
 }  
  
 @Scheduled(cron = "0/5 \* \* \* \* \*")  
 public void cleanDirectory() throws IOException {  
 *log*.info("cleanDirectory started");  
  
 if(Files.*exists*(orderSrc)){  
 Files.*delete*(orderSrc);  
 }  
 if(Files.*exists*(itemSrc)){  
 Files.*delete*(itemSrc);  
 }  
  
 List<Item> items = itemService.findAll();  
 List<Order> orders = orderService.findAll();  
  
 StringBuilder ordersData = new StringBuilder();  
 StringBuilder itemsData = new StringBuilder();  
  
  
 for (Order order : orders){  
 ordersData.append(" ")  
 .append(order.getId())  
 .append(" ")  
 .append(order.getOrderDate())  
 .append(" ")  
 .append(order.getItems())  
 .append("\n");  
 }  
 for (Item item : items){  
 itemsData.append(item.getId())  
 .append(" ")  
 .append(item.getName())  
 .append(" ")  
 .append(item.getPrice())  
 .append(" ")  
 .append(item.getCreationDate())  
 .append("\n");  
 }  
 FileWriter orderWriter = new FileWriter("/Users/tweazy/Desktop/order.txt");  
 FileWriter itemWriter = new FileWriter("/Users/tweazy/Desktop/item.txt");  
  
 orderWriter.write(ordersData.toString());  
 orderWriter.close();  
  
 itemWriter.write(itemsData.toString());  
 itemWriter.close();  
 }  
}

Листинг 77 – код для пр. работы №22

package com.example.javaframeworktemplates.task22.jmx;  
  
import java.io.IOException;  
  
public interface SchedulerMBean {  
 void backupData() throws IOException;  
}

Листинг 78 – код для пр. работы №22

package com.example.javaframeworktemplates.task22.jmx.impl;  
  
import com.example.javaframeworktemplates.task22.jmx.SchedulerMBean;  
import com.example.javaframeworktemplates.task22.services.SchedulerService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.jmx.export.annotation.ManagedAttribute;  
import org.springframework.jmx.export.annotation.ManagedResource;  
import org.springframework.stereotype.Component;  
  
import java.io.IOException;  
  
@Component  
@ManagedResource(description = "Create backup")  
public class SchedulerMBeanImpl implements SchedulerMBean {  
 private final SchedulerService schedulerService;  
  
 @Autowired  
 public SchedulerMBeanImpl(SchedulerService schedulerService){  
 this.schedulerService = schedulerService;  
 }  
  
 @Override  
 public void backupData() throws IOException {  
 schedulerService.cleanDirectory();  
 }  
}

1. **Результат работы программы**

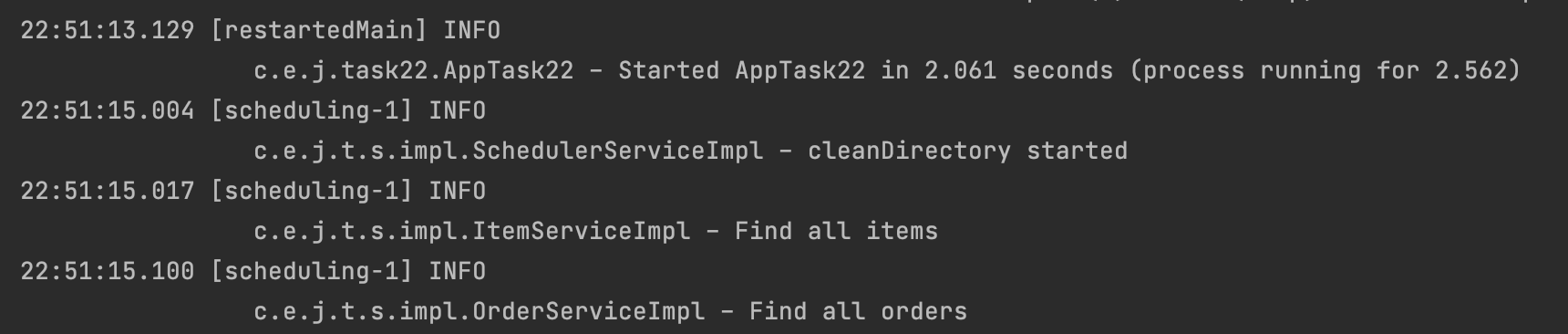
****

Рисунок 32 – Результат работы программы

**Практическая работа №23**

1. **Постановка задачи и персональный вариант**

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

Постановка задачи: 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.

1. **Код и пояснение к коду**

Листинг 79 – код для пр. работы №23

package com.example.javaframeworktemplates.task23.security.custom.models;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
import lombok.RequiredArgsConstructor;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
  
import java.util.Collection;  
import java.util.Set;  
  
@Data  
@Entity  
@AllArgsConstructor  
@NoArgsConstructor  
@Table(name = "users")  
public class User implements UserDetails {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 private String username;  
 private String firstname;  
 private String lastname;  
 private String email;  
 private String password;  
 @ManyToMany(fetch = FetchType.*EAGER*, cascade = {CascadeType.*REFRESH*})  
 private Set<Role> roles;  
 public User(String username,String email, String password, Set<Role> roles) {  
 this.username = username;  
 this.email = email;  
 this.password = password;  
 this.roles = roles;  
 }  
  
 public User(String username, String firstname, String lastname, String email, String password) {  
 this.username = username;  
 this.firstname = firstname;  
 this.lastname = lastname;  
 this.email = email;  
 this.password = password;  
 }  
  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
 return getRoles();  
 }  
  
 @Override  
 public boolean isAccountNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isAccountNonLocked() {  
 return true;  
 }  
  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isEnabled() {  
 return true;  
 }  
}

Листинг 80 – код для пр. работы №23

package com.example.javaframeworktemplates.task23.security.custom.models;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
import org.springframework.security.core.GrantedAuthority;  
  
import java.util.Set;  
  
@Data  
@Entity  
@AllArgsConstructor  
@NoArgsConstructor  
@Table(name = "roles")  
public class Role implements GrantedAuthority {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 private String name;  
 @ManyToMany(mappedBy = "roles", fetch = FetchType.*EAGER*, cascade = {CascadeType.*REFRESH*})  
 @Transient  
 private Set<User> users;  
  
 public Role(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public String getAuthority() {  
 return getName();  
 }  
}

Листинг 81 – код для пр. работы №23

package com.example.javaframeworktemplates.task23.security.custom.services.impl;  
  
import com.example.javaframeworktemplates.task23.security.custom.models.Role;  
import com.example.javaframeworktemplates.task23.security.custom.repositories.RoleRepository;  
import com.example.javaframeworktemplates.task23.security.custom.services.RoleService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
@Service  
public class RoleServiceImpl implements RoleService {  
 private final RoleRepository roleRepository;  
 @Autowired  
 public RoleServiceImpl(RoleRepository roleRepository) {  
 this.roleRepository = roleRepository;  
 }  
  
 @Override  
 public Role findByName(String name) {  
 return roleRepository.findByName(name);  
 }  
  
 @Override  
 public Role saveRole(Role role) {  
 return roleRepository.save(role);  
 }  
  
 @Override  
 public List<Role> findAllRoles() {  
 return roleRepository.findAll();  
 }  
  
 @Override  
 public List<Role> saveAllRoles(Iterable<Role> roles) {  
 return roleRepository.saveAll(roles);  
 }  
}

Листинг 82 – код для пр. работы №23

package com.example.javaframeworktemplates.task23.security.custom.services.impl;  
  
  
import com.example.javaframeworktemplates.task23.security.custom.models.User;  
import com.example.javaframeworktemplates.task23.security.custom.models.enumerates.ERole;  
import com.example.javaframeworktemplates.task23.security.custom.repositories.UserRepository;  
import com.example.javaframeworktemplates.task23.security.custom.services.RoleService;  
import com.example.javaframeworktemplates.task23.security.custom.services.UserService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.stereotype.Service;  
  
import java.util.Collections;  
import java.util.List;  
  
  
@Service  
public class UserServiceImpl implements UserService {  
 private final UserRepository userRepository;  
 private final RoleService roleService;  
 private final PasswordEncoder passwordEncoder;  
  
 @Autowired  
 public UserServiceImpl(  
 UserRepository userRepository,  
 RoleService roleService,  
 PasswordEncoder passwordEncoder  
 ) {  
 this.userRepository = userRepository;  
 this.roleService = roleService;  
 this.passwordEncoder = passwordEncoder;  
 }  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 User user = userRepository.findByUsername(username);  
 if (user == null) {  
 throw new UsernameNotFoundException("User not found");  
 }  
 return user;  
 }  
  
 @Override  
 public User findByUsername(String username) {  
 return userRepository.findByUsername(username);  
 }  
  
 @Override  
 public User saveUser(User user) {  
 if (userRepository.findByUsername(user.getUsername()) != null) {  
 return null;  
 }  
 if (user.getRoles() == null || user.getRoles().size() == 0) {  
 user.setRoles(Collections.*singleton*(roleService.findByName(ERole.*ROLE\_USER*.toString())));  
 }  
 user.setPassword(passwordEncoder.encode(user.getPassword()));  
 return userRepository.save(user);  
 }  
  
 @Override  
 public List<User> findAllUsers() {  
 return userRepository.findAll();  
 }  
  
 @Override  
 public List<User> saveAllUsers(Iterable<User> users) {  
 return userRepository.saveAll(users);  
 }  
}

Листинг 83 – код для пр. работы №23

package com.example.javaframeworktemplates.task23.security.controllers;  
  
import com.example.javaframeworktemplates.task23.security.custom.models.User;  
import com.example.javaframeworktemplates.task23.security.custom.services.UserService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
  
@Controller  
public class AuthController {  
 private final UserService userService;  
  
 @Autowired  
 public AuthController(UserService userService) {  
 this.userService = userService;  
 }  
  
 @GetMapping("/register")  
 public String register() {  
 return "views/register";  
 }  
  
 @PostMapping("/register")  
 public String registerPost(  
 @RequestParam("username") String username,  
 @RequestParam("firstname") String firstname,  
 @RequestParam("lastname") String lastname,  
 @RequestParam("email") String email,  
 @RequestParam("password") String password  
 ) {  
 userService.saveUser(new User(  
 username,  
 firstname,  
 lastname,  
 email,  
 password  
 ));  
  
 return "redirect:login";  
 }  
}

Листинг 84 – код для пр. работы №23

package com.example.javaframeworktemplates.task23.security.configs;  
  
import com.example.javaframeworktemplates.task23.security.custom.models.Role;  
import com.example.javaframeworktemplates.task23.security.custom.models.User;  
import com.example.javaframeworktemplates.task23.security.custom.models.enumerates.ERole;  
import com.example.javaframeworktemplates.task23.security.custom.services.RoleService;  
import com.example.javaframeworktemplates.task23.security.custom.services.UserService;  
import jakarta.annotation.PostConstruct;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.config.Customizer;  
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.web.SecurityFilterChain;  
  
import java.util.ArrayList;  
import java.util.HashSet;  
import java.util.List;  
  
@Slf4j  
@Configuration  
@EnableWebSecurity  
public class SecurityConfig {  
 private final UserDetailsService userDetailsService;  
 private final PasswordEncoder passwordEncoder;  
 private final UserService userService;  
 private final RoleService roleService;  
  
 @Autowired  
 public SecurityConfig(  
 UserDetailsService userDetailsService,  
 PasswordEncoder passwordEncoder,  
 UserService userService,  
 RoleService roleService  
 ) {  
 this.userDetailsService = userDetailsService;  
 this.passwordEncoder = passwordEncoder;  
 this.userService = userService;  
 this.roleService = roleService;  
 }  
  
 @Autowired  
 public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {  
 auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder);  
 }  
  
 @Bean  
 public UserDetailsService userDetailsService() {  
 return userDetailsService;  
 }  
  
 @Bean  
 public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
 http  
 .cors(AbstractHttpConfigurer::disable)  
 .csrf(AbstractHttpConfigurer::disable)  
 .authorizeHttpRequests(authorize -> {  
 try {  
 authorize  
 .requestMatchers("/login").permitAll()  
 .requestMatchers("/register").permitAll()  
 .anyRequest().authenticated()  
 .and()  
 .formLogin(Customizer.*withDefaults*())  
 .logout(Customizer.*withDefaults*());  
 } catch (Exception e) {  
 throw new RuntimeException(e);  
 }  
 }  
 );  
 return http.build();  
 }  
  
 @PostConstruct  
 public void initialization() {  
 List<Role> roles = roleService.saveAllRoles(new ArrayList<>() {{  
 add(new Role(ERole.*ROLE\_ADMIN*.toString()));  
 add(new Role(ERole.*ROLE\_USER*.toString()));  
 add(new Role(ERole.*ROLE\_PUBLISHER*.toString()));  
 }});  
 User admin = userService.saveUser(new User(  
 "admin",  
 "admin@mail.ru",  
 "password",  
 new HashSet<>(roleService.findAllRoles())  
 ));  
  
 *log*.info("roles: {}, admin: {}", roles, admin);  
 }  
}

1. **Результат работы программы**

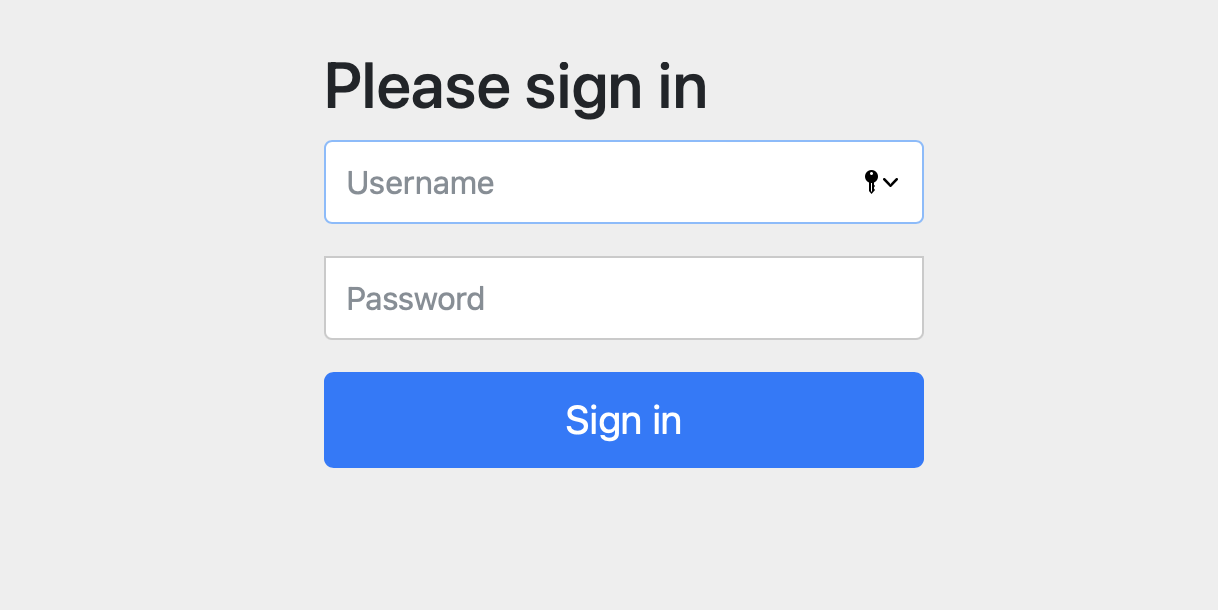
****

Рисунок 33 – Результат работы программы

**Практическая работа №24**

1. **Постановка задачи и персональный вариант**

Цель работы: Тестирование в Spring Framework с использованием Junit.

Постановка задачи: Написать модульное тестирование для всех классов сервисов приложения из предыдущего задания.

1. **Код и пояснение к коду**

Листинг 85 – код для пр. работы №24

package com.example.javaframeworktemplates;  
  
import com.example.javaframeworktemplates.task23.entities.Item;  
import com.example.javaframeworktemplates.task23.repository.ItemRepository;  
import com.example.javaframeworktemplates.task23.services.EmailService;  
import com.example.javaframeworktemplates.task23.services.impl.ItemServiceImpl;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.junit.jupiter.MockitoExtension;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
import static org.mockito.ArgumentMatchers.*anyString*;  
import static org.mockito.ArgumentMatchers.*eq*;  
import static org.mockito.Mockito.*verify*;  
import static org.mockito.Mockito.*when*;  
  
@ExtendWith(MockitoExtension.class)  
public class ItemServiceImplTest {  
  
 @InjectMocks  
 private ItemServiceImpl itemService;  
  
 @Mock  
 private ItemRepository itemRepository;  
  
 @Mock  
 private EmailService emailService;  
  
 @Test  
 public void findAll(){  
 Item item = new Item();  
 item.setId(35);  
 item.setName("item1");  
 item.setPrice(300);  
 item.setCreationDate(LocalDate.*of*(2023, 5, 30));  
  
 List<Item> items = new ArrayList<>();  
 items.add(item);  
  
 *when*(itemRepository.findAll()).thenReturn(items);  
  
 List<Item> retrievedItems = itemService.findAll();  
 *assertEquals*(items, retrievedItems);  
 }  
  
 @Test  
 public void deleteItem() {  
 Item item = new Item();  
 item.setId(35);  
 item.setName("item1");  
 item.setPrice(300);  
 item.setCreationDate(LocalDate.*of*(2023, 5, 30));  
  
 itemService.saveItem(item);  
  
 ResponseEntity<?> deleteResponse = itemService.deleteItem(item);  
  
 *verify*(emailService).sendEmail(*eq*("zhavoronkov03@inbox.ru") ,*eq*("saveItem"), *anyString*());  
 *assertEquals*(HttpStatus.*NOT\_FOUND*, deleteResponse.getStatusCode());  
 }  
 @Test  
 public void findById(){  
 Item item = new Item();  
 item.setId(35);  
 item.setName("item1");  
 item.setPrice(300);  
 item.setCreationDate(LocalDate.*of*(2023, 5, 30));  
  
 *when*(itemRepository.findById(35)).thenReturn(Optional.*of*(item));  
  
 ResponseEntity<Item> responseEntity = itemService.findById(35);  
  
  
 *assertEquals*(item, responseEntity.getBody());  
  
  
 }  
   
}

Листинг 86 – код для пр. работы №24

package com.example.javaframeworktemplates;  
  
import com.example.javaframeworktemplates.task23.entities.Item;  
import com.example.javaframeworktemplates.task23.entities.Order;  
import com.example.javaframeworktemplates.task23.repository.OrderRepository;  
import com.example.javaframeworktemplates.task23.services.EmailService;  
import com.example.javaframeworktemplates.task23.services.impl.OrderServiceImpl;  
import org.aspectj.weaver.ast.Or;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.junit.jupiter.MockitoExtension;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.mockito.ArgumentMatchers.*anyString*;  
import static org.mockito.ArgumentMatchers.*eq*;  
import static org.mockito.Mockito.*verify*;  
import static org.mockito.Mockito.*when*;  
  
@ExtendWith(MockitoExtension.class)  
public class OrderServiceImplTest {  
  
 @InjectMocks  
 private OrderServiceImpl orderService;  
  
 @Mock  
 private OrderRepository orderRepository;  
  
 @Mock  
 private EmailService emailService;  
  
 @Test  
 public void findAll(){  
 Order order = new Order();  
 order.setId(35);  
 order.setOrderDate(LocalDate.*of*(2023, 5, 30));  
  
 List<Order> orders = new ArrayList<>();  
 orders.add(order);  
  
 *when*(orderRepository.findAll()).thenReturn(orders);  
  
 List<Order> retrievedOrders = orderService.findAll();  
 *assertEquals*(orders, retrievedOrders);  
 }  
  
 @Test  
 public void deleteOrder(){  
 Order order = new Order();  
 order.setId(35);  
 order.setOrderDate(LocalDate.*of*(2023, 5, 30));  
  
 orderService.saveOrder(order);  
  
 ResponseEntity<?> deleteResponse = orderService.deleteOrder(order);  
  
 *verify*(emailService).sendEmail(*eq*("zhavoronkov03@inbox.ru") ,*eq*("saveOrder"), *anyString*());  
 *assertEquals*(HttpStatus.*NOT\_FOUND*, deleteResponse.getStatusCode());  
 }  
  
 @Test  
 public void findById(){  
 Order order = new Order();  
 order.setId(35);  
 order.setOrderDate(LocalDate.*of*(2023, 5, 30));  
  
 *when*(orderRepository.findById(35)).thenReturn(Optional.*of*(order));  
  
 ResponseEntity<Order> responseEntity = orderService.findById(35);  
  
 *assertEquals*(order, responseEntity.getBody());  
  
 }  
}

1. **Результат работы программы**

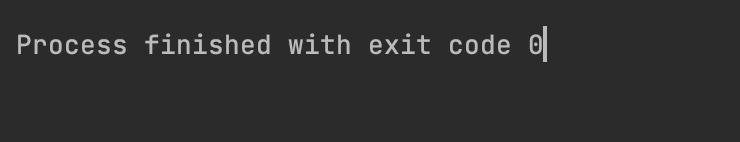
****

Рисунок 34 – Результат работы программы

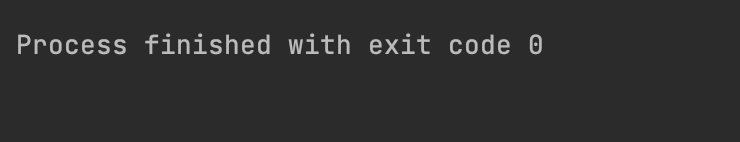
****

Рисунок 35 – Результат работы программы