|  |
| --- |
|  |
| 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** | ***Гаджиев Р.С.*** |
| Принял ассистент кафедры ИиППО | ***Ермаков С.Р.*** |

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

Москва 2024

**Оглавление**

[Практическая работа №1 3](#_Toc164336525)

[Практическая работа №2 4](#_Toc164336526)

[Практическая работа №3 5](#_Toc164336527)

[Практическая работа №4 7](#_Toc164336528)

[Практическая работа №5 9](#_Toc164336529)

[Практическая работа №6 11](#_Toc164336530)

[Практическая работа №7 18](#_Toc164336531)

[Практическая работа №8 22](#_Toc164336532)

[Практическая работа №9 24](#_Toc164336533)

[Практическая работа №10 26](#_Toc164336534)

[Практическая работа №11 29](#_Toc164336535)

[Практическая работа №12 31](#_Toc164336536)

[Практическая работа №13 33](#_Toc164336537)

[Практическая работа №14 35](#_Toc164336538)

[Практическая работа №15 39](#_Toc164336539)

[Практическая работа №16 44](#_Toc164336540)

[Практическая работа №17 51](#_Toc164336541)

[Практическая работа №18 56](#_Toc164336542)

[Практическая работа №19 57](#_Toc164336543)

[Практическая работа №20 60](#_Toc164336544)

[Практическая работа №21 61](#_Toc164336545)

[Практическая работа №22 62](#_Toc164336546)

# **Практическая работа №1**

1. **Постановка задачи и персональный вариант**

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

Постановка задачи: имплементировать интерфейс Comparator, сравнивающий два массива с одинаковыми типами элементов по количеству элементов в данных массивах.

1. **Код**

Файл: Main.java

package Task1;

import java.util.Comparator;

public class Main

{

public static void main(String[] args)

{

Comparator<Object[]> comparator = Comparator.comparingInt(arr -> arr.length);

var newArr1 = new Integer[]{1, 2, 4, 7};

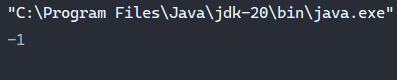
var newArr2 = new Integer[]{1, 10, 10, 10, 10};

System.out.println(comparator.compare(newArr1, newArr2));

}

}

1. **Результат работы программы**



# **Практическая работа №2**

1. **Постановка задачи и персональный вариант**

Тема: Работа со Stream API в Java 8.

Постановка задачи: фильтрация по возрасту больше, чем 20, сортировка по последней букве имени, увеличение возраста каждого на 3, вычисление среднего возраста всех элементов.

1. **Код**

Файл: Main.java

package Task1;

import java.util.Comparator;

public class Main

{

public static void main(String[] args)

{

Comparator<Object[]> comparator = Comparator.comparingInt(arr -> arr.length);

var newArr1 = new Integer[]{1, 2, 4, 7};

var newArr2 = new Integer[]{1, 10, 10, 10, 10};

System.out.println(comparator.compare(newArr1, newArr2));

}

}  
  
class HumanComparator implements Comparator<Human>  
{  
 @Override  
 public int compare(Human x1, Human x2) {  
 return x1.firstName.substring(x1.firstName.length() - 1)  
 .compareTo(x2.firstName.substring(x2.firstName.length() - 1));  
 }  
}

1. **Результат работы программы**



# **Практическая работа №3**

1. **Постановка задачи и персональный вариант**

Тема: знакомство с конкурентным программированием в Java. Потокобезопасность, ключевое слово syncrhonized, мьютексы, семафоры, мониторы, барьеры.

Постановка задачи: создать свои потокобезопасные имплементации интерфейсов: List с использованием Semaphore и Map с использованием ключевого слова synchronized.

1. **Код**

Файл: Main.java

package Task3.Task3\_1;

import java.util.ArrayList;

import java.util.concurrent.Semaphore;

public class Main

{

public static void main(String[] args)

{

// List с использованием Semaphore

var semaphore = new Semaphore(1);

var list = new ArrayList<Integer>();

var writer1 = new Thread(() -> {

try

{

semaphore.acquire();

list.add(1);

list.add(2);

list.add(3);

list.add(4);

System.out.println("Список 1 " + list);

} catch (InterruptedException e)

{

throw new RuntimeException(e);

}

finally

{

semaphore.release();

}

});

var writer2 = new Thread(() -> {

try

{

semaphore.acquire();

list.add(5);

list.add(6);

list.add(7);

list.add(8);

System.out.println("Список 2 " + list);

} catch (InterruptedException e)

{

throw new RuntimeException(e);

}

finally

{

semaphore.release();

}

});

writer1.start();

writer2.start();

}

}

Файл: Main.java

package Task3.Task3\_2;

import java.util.HashMap;

public class Main

{

public static HashMap<String, Integer> synchronizedMap = new HashMap<>();

public static int iterator = 0;

public static void main(String[] args)

{

// Map с использованием ключевого слова synchronized

for (int i = 1; i < 5; i++)

{

var t = new Thread(() -> {

synchronized (synchronizedMap) {

for (int j = iterator; j < iterator + 4; j++)

{

synchronizedMap.put("key" + j, j);

System.out.println(Thread.currentThread().getName() + " " + j);

}

iterator+=4;

}

});

t.setName("Thread" + i);

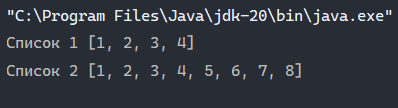
t.start();

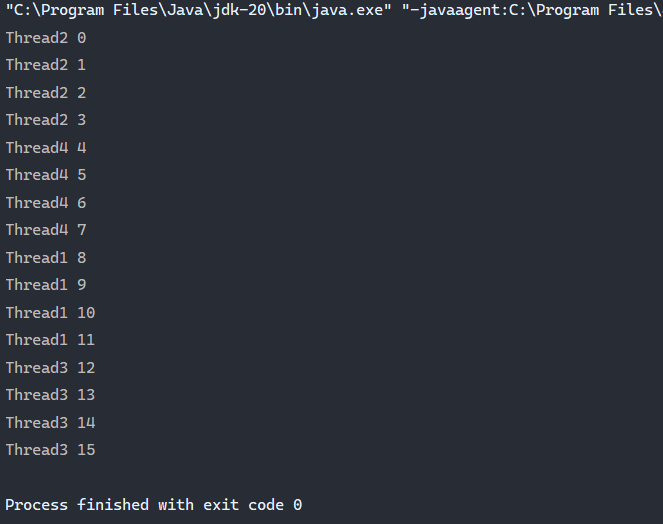
}

}

}

1. **Результат работы программы**





# **Практическая работа №4**

1. **Постановка задачи и персональный вариант**

Тема: работа с ExecutorService, CompletableFuture.

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

1. **Код**

Файл: Main.java

package Task4;

import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.BlockingQueue;

import java.util.concurrent.LinkedBlockingQueue;

public class Main

{

public static void main(String[] args)

{

// Пример использования

CustomExecutorService executorService = new CustomExecutorService(5);

for (int i = 1; i < 6; i++)

{

final int taskNumber = i;

executorService.submit(() -> {

System.out.println("Task " + taskNumber + " executed by thread " + Thread.currentThread().getName());

});

}

executorService.shutdown();

}

}

class CustomExecutorService

{

private final List<WorkerThread> threads;

private final BlockingQueue<Runnable> taskQueue;

public CustomExecutorService(int threadPoolSize)

{

this.threads = new ArrayList<>(threadPoolSize);

this.taskQueue = new LinkedBlockingQueue<>();

for (int i = 0; i < threadPoolSize; i++)

{

WorkerThread workerThread = new WorkerThread();

threads.add(workerThread);

workerThread.start();

}

}

public void submit(Runnable task)

{

try

{

taskQueue.put(task);

}

catch (InterruptedException e)

{

Thread.currentThread().interrupt();

}

}

public void shutdown()

{

for (WorkerThread workerThread : threads)

{

workerThread.interrupt();

}

}

private class WorkerThread extends Thread

{

@Override

public void run()

{

while (!Thread.currentThread().isInterrupted())

{

try

{

Runnable task = taskQueue.take();

task.run();

}

catch (InterruptedException e)

{

Thread.currentThread().interrupt();

}

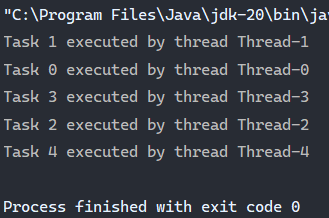
}

}

}

}

1. **Результат работы программы**



# **Практическая работа №5**

1. **Постановка задачи и персональный вариант**

Тема: 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**

Файл: Main.java

package Task5;

public class Main

{

public static void main(String[] args)

{

// Реализовать паттерн Singleton как минимум 3-мя способами

System.out.println(Singleton1.getInstance());

System.out.println(Singleton2.getInstance());

for (int i = 1; i < 4; i++)

{

var t = new Thread(() -> {

System.out.println(Singleton3.getInstance() + " " + Thread.currentThread().getName());

});

t.setName("Поток " + i);

t.start();

}

}

}

// Прстой пример

class Singleton1

{

private static final Singleton1 Instance = new Singleton1();

private Singleton1() { }

public static Singleton1 getInstance()

{

return Instance;

}

}

// Ленивая инициализация

class Singleton2

{

private static Singleton2 Instance;

private Singleton2() { }

public static Singleton2 getInstance()

{

if (Instance == null)

{

Instance = new Singleton2();

}

return Instance;

}

}

// Потокобезопасный

class Singleton3

{

private Singleton3() { }

private static final class InstanceHolder

{

private static final Singleton3 Instance = new Singleton3();

}

public static synchronized Singleton3 getInstance()

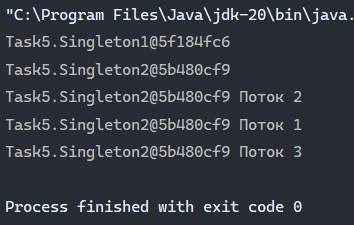
{

return InstanceHolder.Instance;

}

}

1. **Результат работы программы**



# **Практическая работа №6**

1. **Постановка задачи и персональный вариант**

Тема: знакомство с реализацией порождающих паттернов проектирования.

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

1. **Код**

Файл: Main.java

package Task6;

public class Main

{

public static void main(String[] args)

{

// «Фабричный метод»

System.out.println("Паттерн «Фабричный метод»:");

new FactoryMethod();

System.out.println();

// «Абстрактная фабрика»

System.out.println("Паттерн «Абстрактная фабрика»:");

new AbstractFactory();

System.out.println();

// «Строитель»

System.out.println("Паттерн «Строитель»:");

new Builder();

System.out.println();

// «Прототип»

System.out.println("Паттерн «Прототип»:");

new Prototype();

}

}

Файл: FactoryMethod.java

package Task6;

public class FactoryMethod

{

public FactoryMethod()

{

Work();

}

private void Work()

{

Factory factory = new PanelFactory("Завод панельных домов");

factory.BuildHouse();

factory = new WoodFactory("Завод деревянных домов");

factory.BuildHouse();

}

}

// Фабрики

abstract class Factory

{

public String Name;

public Factory(String name)

{

Name = name;

}

abstract House BuildHouse();

}

class PanelFactory extends Factory

{

public PanelFactory(String name)

{

super(name);

}

@Override

House BuildHouse()

{

return new PanelHouse();

}

}

class WoodFactory extends Factory

{

public WoodFactory(String name)

{

super(name);

}

@Override

House BuildHouse()

{

return new WoodHouse();

}

}

// Виды домов

interface House { }

class PanelHouse implements House

{

public PanelHouse()

{

System.out.println("Panel house!");

}

}

class WoodHouse implements House

{

public WoodHouse()

{

System.out.println("Wooden house!");

}

}

Файл: AbstractFactory.java

package Task6;

public class AbstractFactory

{

public AbstractFactory()

{

Work();

}

private void Work()

{

var hero1 = new Hero(new ElfFactory());

var hero2 = new Hero(new TankFactory());

hero1.Attack();

hero2.Attack();

}

}

// Оружие

abstract class Weapon

{

abstract void Hit();

}

class Sword extends Weapon

{

public void Hit()

{

System.out.println("Удар мечом!");

}

}

class Bow extends Weapon

{

public void Hit()

{

System.out.println("Выстрел из лука!");

}

}

// Герой

class Hero

{

private Weapon HeroWeapon;

public Hero(HeroFactory factory)

{

HeroWeapon = factory.CreateWeapon();

}

public void Attack()

{

HeroWeapon.Hit();

}

}

// Фабрики

abstract class HeroFactory

{

abstract Weapon CreateWeapon();

}

class TankFactory extends HeroFactory

{

public Weapon CreateWeapon()

{

return new Sword();

}

}

class ElfFactory extends HeroFactory

{

public Weapon CreateWeapon()

{

return new Bow();

}

}

Файл: Builder.java

package Task6;

public class Builder

{

public Builder()

{

Work();

}

private void Work()

{

var baker = new Baker();

var bread1 = baker.CreateBread(new RyeBreadBuilder());

System.out.println(bread1);

var bread2 = baker.CreateBread(new WheatBreadBuilder());

System.out.println(bread2);

}

}

// Для хлеба

class Flour

{

public String FlourType;

public Flour(String flourType)

{

FlourType = flourType;

}

}

class Additive

{

public String AdditiveName;

public Additive(String additiveName)

{

AdditiveName = additiveName;

}

}

class Salt

{

public int saltCount;

public Salt(int count)

{

saltCount = count;

}

}

class Bread

{

public String name;

public Flour flour;

public Additive additive;

public Salt salt;

@Override

public String toString()

{

return "Хлеб " + name + " {" +

"мука = " + flour.FlourType +

", добавки = " + additive.AdditiveName +

", соль = " + salt.saltCount + "}";

}

}

// Получение хлеба

class Baker

{

public Bread CreateBread(BreadBuilder builder)

{

builder.SetName();

builder.SetFlour();

builder.SetAdditive();

builder.SetSalt();

return builder.bread;

}

}

abstract class BreadBuilder

{

public Bread bread;

public BreadBuilder()

{

bread = new Bread();

}

abstract void SetName();

abstract void SetFlour();

abstract void SetAdditive();

abstract void SetSalt();

}

class RyeBreadBuilder extends BreadBuilder

{

void SetName()

{

this.bread.name = "Ржаной";

}

void SetFlour()

{

this.bread.flour = new Flour("Ржанная мука 1 сорт");

}

void SetAdditive()

{

this.bread.additive = new Additive("Отсутствуют");

}

void SetSalt()

{

this.bread.salt = new Salt(4);

}

}

class WheatBreadBuilder extends BreadBuilder

{

void SetName()

{

this.bread.name = "Пшеничный";

}

void SetFlour()

{

this.bread.flour = new Flour("Пшеничная мука 1 сорт");

}

void SetAdditive()

{

this.bread.additive = new Additive("Улучшитель хлебопекарный");

}

void SetSalt()

{

this.bread.salt = new Salt(5);

}

}

Файл: Prototype.java

p

package Task6;

public class Prototype

{

public Prototype()

{

Work();

}

private void Work()

{

var figure1 = new Rectangle(20, 40);

var figure1\_1 = figure1.Clone();

figure1.GetInfo();

figure1\_1.GetInfo();

var figure2 = new Circle(5);

var figure2\_1 = figure2.Clone();

figure2.GetInfo();

figure2\_1.GetInfo();

}

}

interface IFigure

{

IFigure Clone();

void GetInfo();

}

class Rectangle implements IFigure

{

private int X;

private int Y;

public Rectangle(int x, int y)

{

X = x;

Y = y;

}

public IFigure Clone()

{

return new Rectangle(this.X, this.Y);

}

public void GetInfo()

{

System.out.println("Прямоугольник: X = " + X + ", Y = " + Y);

}

}

class Circle implements IFigure

{

private int Radius;

public Circle(int radius)

{

Radius = radius;

}

public IFigure Clone()

{

return new Circle(this.Radius);

}

public void GetInfo()

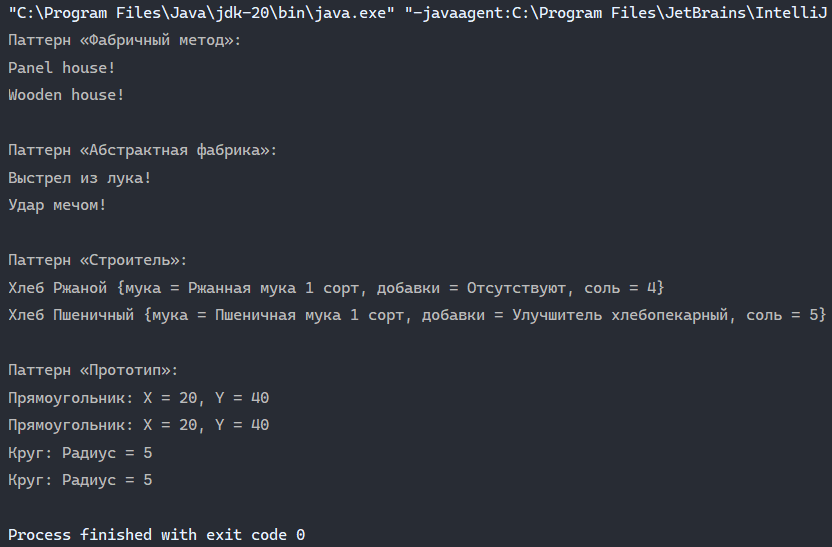
{

System.out.println("Круг: Радиус = " + Radius);

}

}

1. **Результат работы программы**



# **Практическая работа №7**

1. **Постановка задачи и персональный вариант**

Тема: Реализация структурных паттернов проектирования.

Постановка задачи: Написать реализацию паттернов Адаптер и Декоратор.

1. **Код**

Файл: Main.java

package Task7;

public class Main

{

public static void main(String[] args)

{

System.out.println("Паттерн «Адаптер»:");

new Adapter();

System.out.println();

System.out.println("Паттерн «Декоратор»:");

new Decorator();

System.out.println();

}

}

Файл: Adapter.java

package Task7;

public class Adapter

{

public Adapter()

{

Driver driver = new Driver();

Auto auto = new Auto();

driver.Travel(auto);

Camel camel = new Camel();

ITransport camelTransport = new CamelToTransportAdapter(camel);

driver.Travel(camelTransport);

}

}

class Driver

{

public void Travel(ITransport transport)

{

transport.Drive();

}

}

interface ITransport

{

void Drive();

}

class Auto implements ITransport

{

public void Drive()

{

System.out.println("Машина едет врум врум");

}

}

interface IAnimal

{

void Move();

}

class Camel implements IAnimal

{

public void Move()

{

System.out.println("Верблюд идет");

}

}

class CamelToTransportAdapter implements ITransport

{

Camel camel;

public CamelToTransportAdapter(Camel c)

{

camel = c;

}

public void Drive()

{

camel.Move();

}

}

Файл: Decorator.java

package Task7;

public class Decorator

{

public Decorator()

{

Pizza pizza1 = new ItalianPizza();

pizza1 = new TomatoPizza(pizza1);

System.out.println("Название: " + pizza1.Name);

System.out.println("Стоимость: " + pizza1.GetCost());

pizza1 = new BulgerianPizza();

pizza1 = new CheesePizza(pizza1);

System.out.println("Название: " + pizza1.Name);

System.out.println("Стоимость: " + pizza1.GetCost());

}

}

abstract class Pizza

{

public String Name;

public Pizza(String name)

{

Name = name;

}

abstract int GetCost();

}

class ItalianPizza extends Pizza

{

public ItalianPizza()

{

super("Итальянская пицца");

}

@Override

int GetCost()

{

return 5;

}

}

class BulgerianPizza extends Pizza

{

public BulgerianPizza()

{

super("Болгарская пицца");

}

@Override

int GetCost()

{

return 10;

}

}

abstract class PizzaDecorator extends Pizza

{

protected Pizza PizzaModel;

public PizzaDecorator(String name, Pizza PizzaModel)

{

super(name);

this.PizzaModel = PizzaModel;

}

}

class TomatoPizza extends PizzaDecorator

{

public TomatoPizza(Pizza pizza)

{

super(pizza.Name + " с помидорами", pizza);

}

@Override

int GetCost()

{

return PizzaModel.GetCost() + 5;

}

}

class CheesePizza extends PizzaDecorator

{

public CheesePizza(Pizza pizza)

{

super(pizza.Name + " с сыром", pizza);

}

@Override

int GetCost()

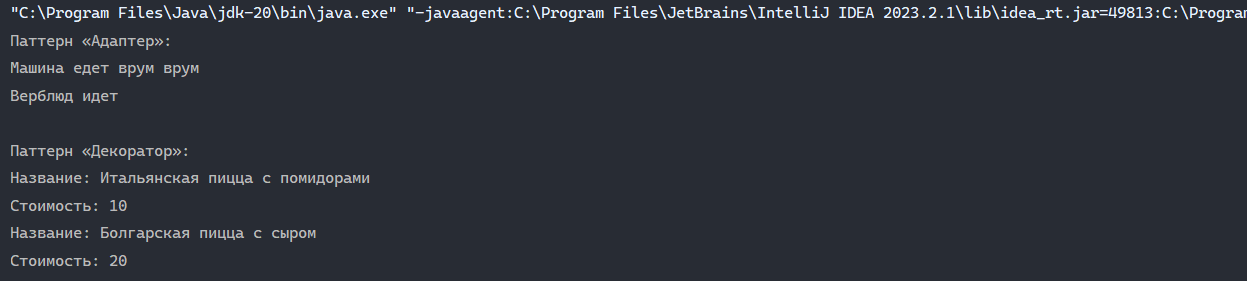
{

return PizzaModel.GetCost() + 10;

}

}

1. **Результат работы программы**



# **Практическая работа №8**

1. **Постановка задачи и персональный вариант**

Тема: Реализация поведенческих паттернов проектирования.

Постановка задачи: Написать реализацию паттернов Стратегия и Шаблонный метод.

1. **Код**

Файл: Main.java

package Task8;

public class Main

{

public static void main(String[] args)

{

System.out.println("Паттерн «Стратегия»:");

new Strategy();

System.out.println();

System.out.println("Паттерн «Шаблонный метод»:");

new Shablon();

System.out.println();

}

}

Файл: Strategy.java

package Task8;

public class Strategy

{

public Strategy()

{

var auto = new Auto("Volvo", new PetrolMove());

auto.Drive();

auto.moveType = new ElectricMove();

auto.Drive();

}

}

interface IMovable

{

void Move();

}

class PetrolMove implements IMovable

{

public void Move()

{

System.out.println("Машина едет на бензине");

}

}

class ElectricMove implements IMovable

{

public void Move()

{

System.out.println("Машина едет на электричестве");

}

}

class Auto

{

String model;

IMovable moveType;

public Auto(String model, IMovable moveType)

{

this.model = model;

this.moveType = moveType;

}

public void Drive()

{

moveType.Move();

}

}

Файл: Shablon.java

package Task8;

public class Shablon

{

public Shablon()

{

var school = new School();

var university = new University();

school.Learn();

university.Learn();

}

}

abstract class Education

{

void Learn()

{

Enter();

DoHomework();

Study();

}

abstract void Enter();

abstract void DoHomework();

abstract void Study();

}

class School extends Education

{

void Enter()

{

System.out.println("Идем в первый класс");

}

void DoHomework()

{

System.out.println("Делаем домашку");

}

void Study()

{

System.out.println("Посещаем уроки");

}

}

class University extends Education

{

void Enter()

{

System.out.println("Поступаем на первый курс");

}

void DoHomework()

{

System.out.println("Делаем домашку (нуу такое)");

}

void Study()

{

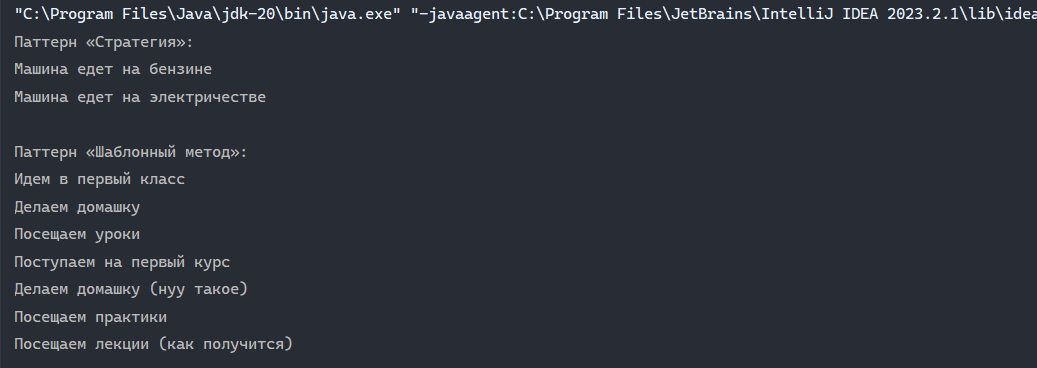
System.out.println("Посещаем практики");

System.out.println("Посещаем лекции (как получится)");

}

}

1. **Результат работы программы**



# **Практическая работа №9**

1. **Постановка задачи и персональный вариант**

Тема: 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**

Файл: src/main/java/ru/mirea/Main.java

package ru.mirea;  
  
public class Main  
{  
 public static void main(String[] args)  
 {  
   
 }  
}

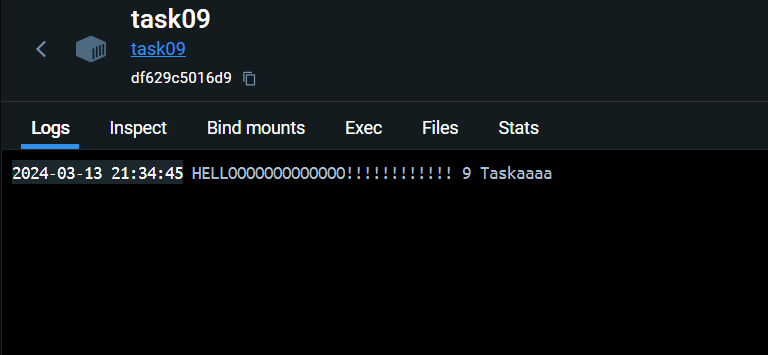
Файл: build.gradle

plugins **{** id 'java'  
**}**group 'ru.mirea'  
  
repositories **{** mavenCentral()  
**}**jar **{** manifest **{** attributes(  
 'Main-Class' : 'ru.mirea.Main'  
 )  
 **}  
}**task copyJar(type: Copy) **{** dependsOn(jar)  
 from file("$buildDir/libs/task09.jar")  
 into file("$rootDir/src/main/resources/")  
**}**task buildImage(type: Exec) **{** dependsOn 'copyJar'  
 commandLine 'docker', 'build', '-t', 'task09', './src/main/resources/'  
**}**task runDocker(type: Exec) **{** dependsOn 'buildImage'  
 commandLine 'docker', 'run', '--name', 'task09', 'task09'  
**}**defaultTasks 'runDocker'

Файл: src/main/resources/Dockerfile

FROM openjdk:15-jdk-alpine  
COPY task09.jar task09.jar  
CMD ["java","-jar","task09.jar"]

1. **Результат работы программы**



# **Практическая работа №10**

1. **Постановка задачи и персональный вариант**

Тема: 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: Интерфейс SortingAlgorithm с методом doSort(), его имплементации: MergeSort, InsertionSort, QuickSort.

1. **Код**

Файл: Task10Application.java

package org.project.task10;  
  
import org.project.task10.config.BeanConfig;  
import org.project.task10.interfaces.SortingAlgorithm;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
  
@SpringBootApplication  
public class Task10Application  
{  
 public static void main(String[] args)  
 {  
 SpringApplication.run(Task10Application.class, args);  
 ApplicationContext context = new AnnotationConfigApplicationContext(BeanConfig.class);  
  
 SortingAlgorithm bean = context.getBean("GetMergeSort", SortingAlgorithm.class);  
 bean.doSort();  
 bean = context.getBean("GetInsertionSort", SortingAlgorithm.class);  
 bean.doSort();  
 bean = context.getBean("GetQuickSort", SortingAlgorithm.class);  
 bean.doSort();  
 }  
}

Файл: interfaces/Task10Application.java

package org.project.task10.interfaces;  
  
public interface SortingAlgorithm  
{  
 void doSort();  
}

Файл: config/BeanConfig.java

package org.project.task10.config;  
  
import org.project.task10.components.\*;  
import org.project.task10.interfaces.SortingAlgorithm;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.ComponentScan;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
@ComponentScan  
public class BeanConfig  
{  
 @Bean  
 public SortingAlgorithm GetMergeSort()  
 {  
 return new MergeSort();  
 }  
  
 @Bean  
 public InsertionSort GetInsertionSort()  
 {  
 return new InsertionSort();  
 }  
  
 @Bean  
 public QuickSort GetQuickSort()  
 {  
 return new QuickSort();  
 }  
}

Файл: components/MergeSort.java

package org.project.task10.components;  
  
import org.project.task10.interfaces.SortingAlgorithm;  
import org.springframework.stereotype.Component;  
  
@Component()  
public class MergeSort implements SortingAlgorithm  
{  
 @Override  
 public void doSort()  
 {  
 System.out.println("Doing merge sort");  
 }  
}

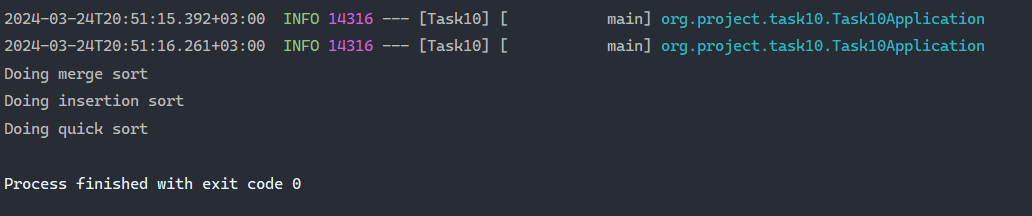
Файл: components/InsertionSort.java

package org.project.task10.components;  
  
import org.project.task10.interfaces.SortingAlgorithm;  
import org.springframework.stereotype.Component;  
  
@Component  
public class InsertionSort implements SortingAlgorithm  
{  
 @Override  
 public void doSort()  
 {  
 System.out.println("Doing insertion sort");  
 }  
}

Файл: components/QuickSort.java

package org.project.task10.components;  
  
import org.project.task10.interfaces.SortingAlgorithm;  
import org.springframework.stereotype.Component;  
  
@Component  
public class QuickSort implements SortingAlgorithm  
{  
 @Override  
 public void doSort()  
 {  
 System.out.println("Doing quick sort");  
 }  
}

1. **Результат работы программы**



# **Практическая работа №11**

1. **Постановка задачи и персональный вариант**

Тема: 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**

Файл: Task11Application.java

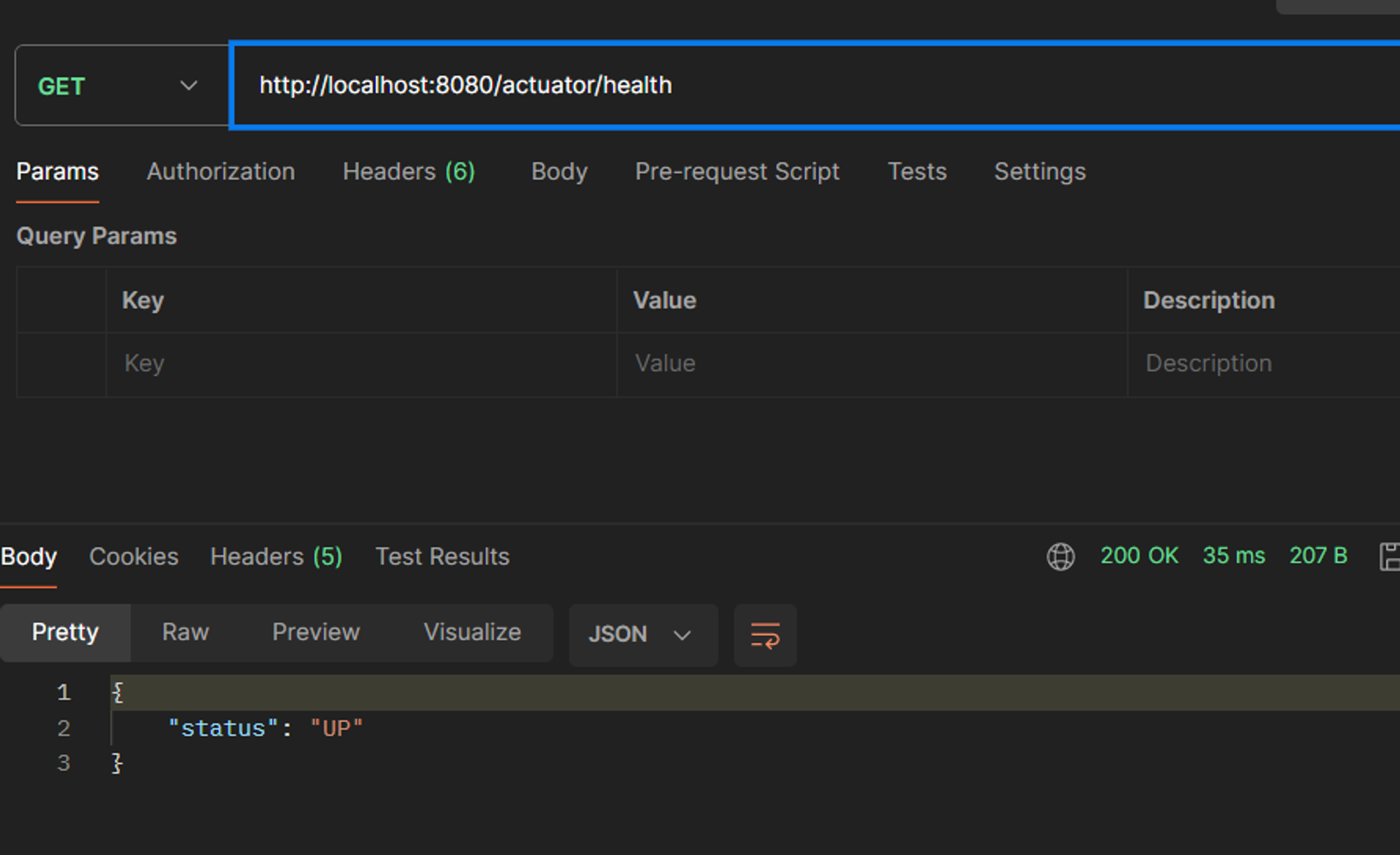
package com.example;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class Task11Application  
{  
 public static void main(String[] args)  
 {  
 SpringApplication.run(Task11Application.class, args);  
 }  
}

Файл: controller/HealthCheckController.java

package com.example.controllers;  
  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class HealthCheckController  
{  
 @GetMapping("/health")  
 public String checkHealth()

{  
 return "Application is up and running!";  
 }  
}

1. **Результат работы программы**



# **Практическая работа №12**

1. **Постановка задачи и персональный вариант**

Тема: 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**

Файл: Task12Application.java

package org.project.task12;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.ApplicationContext;  
  
@SpringBootApplication  
public class Task12Application

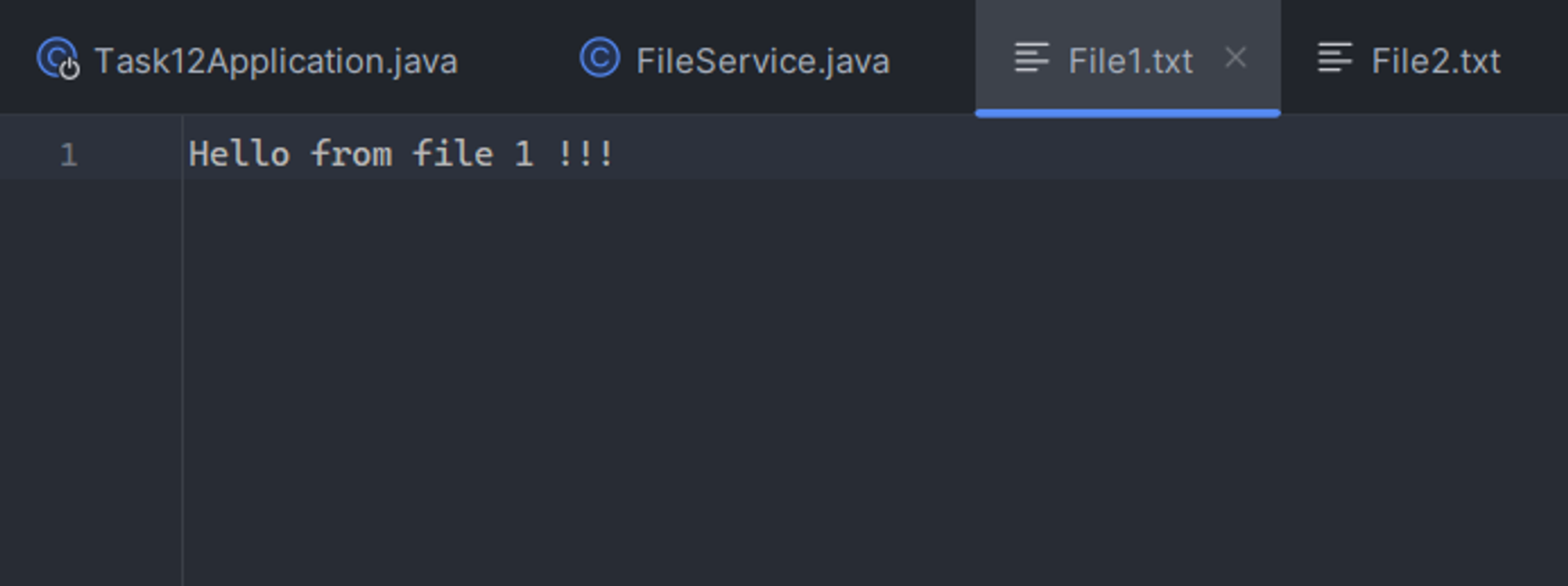
{  
 public static void main(String[] args)  
 {  
 ApplicationContext context = SpringApplication.run(Task12Application.class, args);  
 new FileService();  
 SpringApplication.exit(context);  
 }  
}

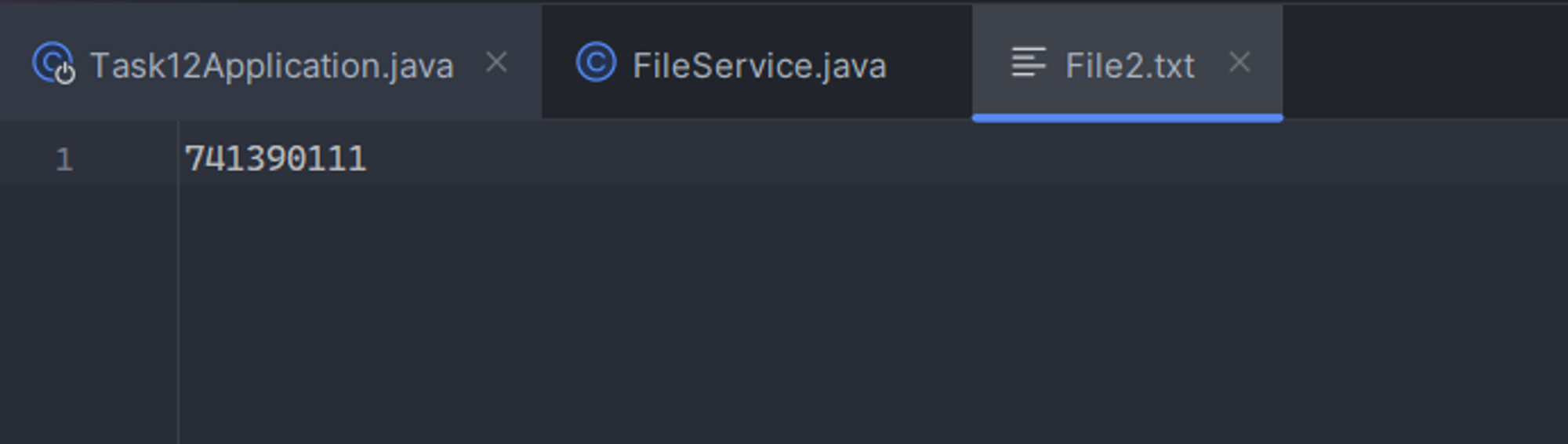
Файл: FileService.java

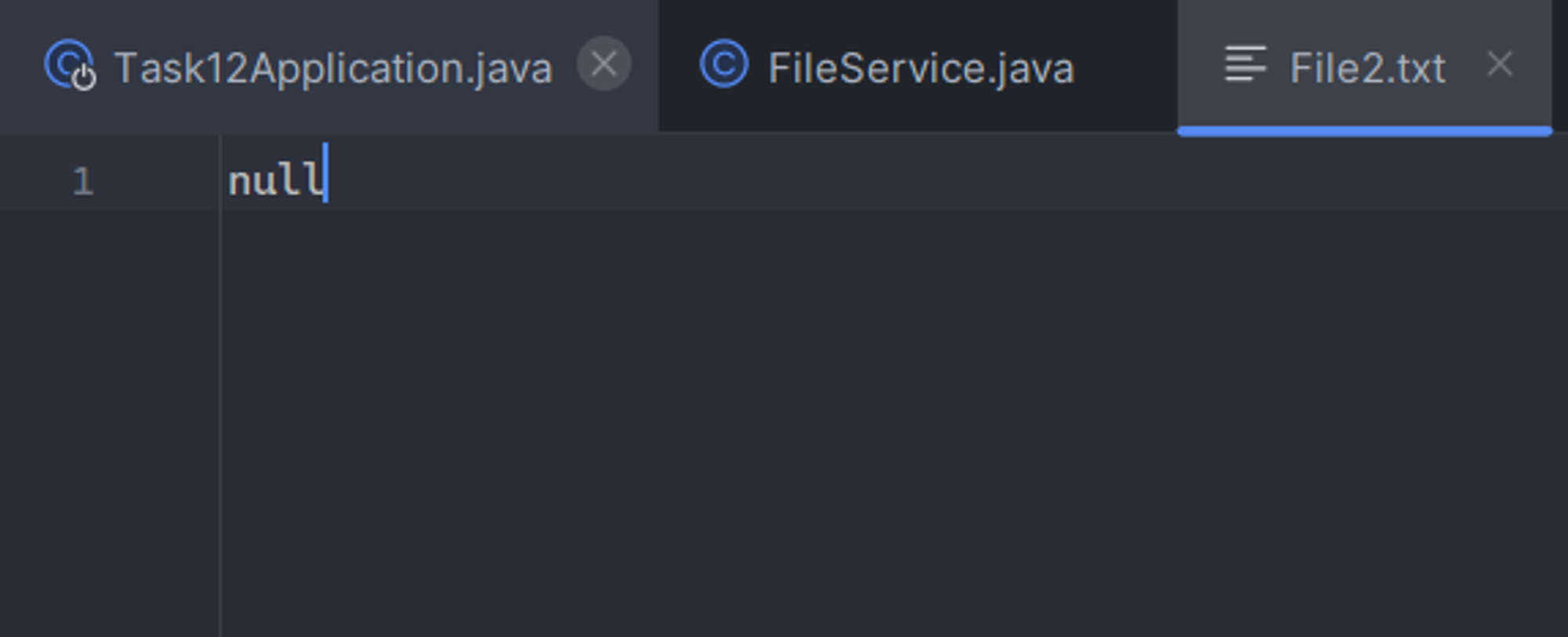
package org.project.task12;  
  
import jakarta.annotation.PostConstruct;  
import jakarta.annotation.PreDestroy;  
import org.springframework.stereotype.Service;  
import java.io.\*;  
  
@Service  
public class FileService  
{  
 String file1 = "D:\\Programming\\JavaPatterns\\Task12\\File1.txt";  
 String file2 = "D:\\Programming\\JavaPatterns\\Task12\\File2.txt";  
  
 @PostConstruct  
 public void PostConstruct() throws IOException  
 {  
 var bufferedWriter = new BufferedWriter(new FileWriter(file2));  
  
 if (new File(file1).exists())  
 {  
 var bufferedReader = new BufferedReader(new FileReader(file1));  
 var strBuilder = new StringBuilder();  
 String line;  
  
 while ((line = bufferedReader.readLine()) != null)  
 {  
 strBuilder.append(line);  
 }

bufferedWriter.write(String.valueOf(strBuilder.hashCode()));  
 System.out.println("Text from 1 hashing to 2");  
 bufferedReader.close();  
 }  
 else  
 {  
 System.out.println("First file does not exist");  
 bufferedWriter.write("null");  
 }  
  
 bufferedWriter.close();  
 }  
  
 @PreDestroy  
 public void PreDestroy()  
 {  
 if(new File(file1).delete())  
 {  
 System.out.println("File 1 - removed");  
 }  
 }  
}

1. **Результат работы программы**



****

****

# 

# **Практическая работа №13**

1. **Постановка задачи и персональный вариант**

Тема: 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**

Файл: Task13Application.java

package org.task13;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class Task13Application {  
 public static void main(String[] args) {  
 SpringApplication.run(Task13Application.class, args);  
 new User();  
 }  
}

Файл: User.java

package org.task13;  
  
import jakarta.annotation.PostConstruct;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
@Component  
public class User  
{  
 @Value("${student.name}")  
 private String firstName;  
 @Value("${student.last\_name}")  
 private String lastName;  
 @Value("${student.group}")  
 private String group;  
  
 @PostConstruct  
 public void print() {  
 System.out.println(firstName + " " + lastName + " " + group);  
 }  
}

Файл: application.properties

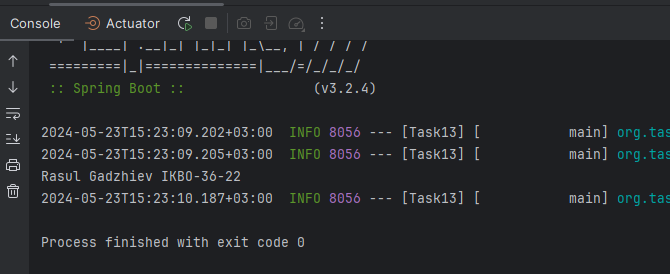
spring.application.name=Task13

student.name=Rasul

student.last\_name=Gadzhiev

student.group=IKBO-36-22

1. **Результат работы программы**



# **Практическая работа №14**

1. **Постановка задачи и персональный вариант**

Тема: Знакомство со Spring MVC. Работа с Rest API в Spring.

Постановка задачи: Создать класс Level с полями complexity, levelName. Создать класс Game с полями name, creationDate. Создать классы-контроллеры для создания, удаления объектов и получения всех объектов каждого типа. Сами объекты хранить в памяти.

1. **Код**

Файл: controller/GameController.java

package org.task14.controller;  
  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.task14.entity.Game;  
  
import java.util.ArrayList;  
import java.util.List;  
  
@RestController  
@RequestMapping("game")  
public class GameController  
{  
 private List<Game> games = new ArrayList<>();

// Add  
 @PostMapping("/add")  
 public ResponseEntity AddGame(@RequestBody Game game)  
 {  
 games.add(game);  
 return ResponseEntity.ok(HttpStatus.OK);  
 }

// Delete  
 @DeleteMapping("/delete/{id}")  
 public ResponseEntity DeleteGame(@PathVariable int id)  
 {  
 games.remove(id);  
 return ResponseEntity.ok(HttpStatus.OK);  
 }

// GetAll  
 @GetMapping()  
 public List<Game> GetAllGame()  
 {  
 return games;  
 }  
}

Файл: controller/LevelController.java

package org.task14.controller;  
  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.task14.entity.Game;  
import org.task14.entity.Level;  
  
import java.util.ArrayList;  
import java.util.List;  
  
@RestController  
@RequestMapping("level")  
public class LevelController  
{  
 private List<Level> levels = new ArrayList<>();  
  
 @PostMapping("/add")  
 public ResponseEntity AddGame(@RequestBody Level level)  
 {  
 levels.add(level);  
 return ResponseEntity.ok(HttpStatus.OK);  
 }  
 // Delete  
 @DeleteMapping("/delete/{id}")  
 public ResponseEntity DeleteGame(@PathVariable int id)  
 {  
 levels.remove(id);  
 return ResponseEntity.ok(HttpStatus.OK);  
 }  
 // GetAll  
 @GetMapping()  
 public List<Level> GetAllGame()  
 {  
 return levels;  
 }  
}

Файл: entity/Game.java

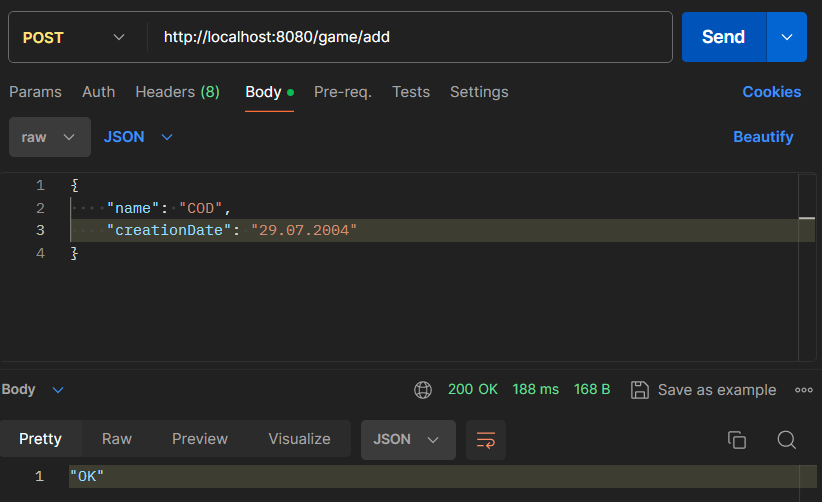
package org.task14.entity;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.Setter;

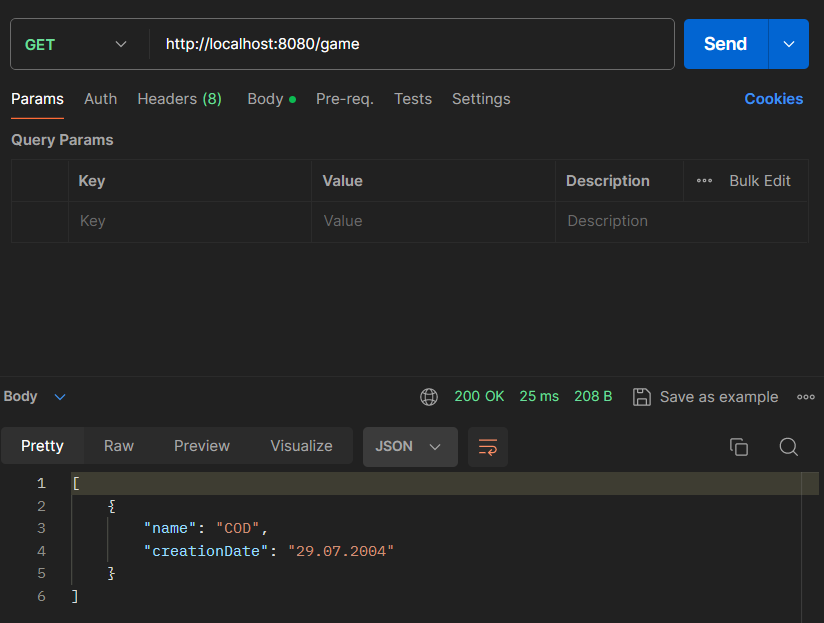
@Getter  
@Setter  
@AllArgsConstructor  
public class Game  
{  
 private String name;  
 private String creationDate;  
}

Файл: entity/Level.java

package org.task14.entity;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.Setter;  
  
@Getter  
@Setter  
@AllArgsConstructor  
public class Level  
{  
 private String complexity;  
 private String levelName;  
}

1. **Результат работы программы**





# **Практическая работа №15**

1. **Постановка задачи и персональный вариант**

Тема: 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**

Файл: entity/Game.java

package org.task15.entity;  
  
import jakarta.persistence.\*;  
import lombok.Getter;  
import lombok.Setter;  
  
@Entity  
@Table(name = "games")  
@Getter  
@Setter  
public class Game  
{  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 @Column(name="name", nullable=false, length=25)  
 private String name;  
 @Column(name="creation\_date", nullable=false, length=25)  
 private String creationDate;  
}

Файл: controller/GameController.java

package org.task15.controller;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.task15.entity.Game;  
import org.task15.service.GameService;  
  
import java.sql.SQLException;  
import java.util.List;  
  
@RestController  
@RequestMapping("api/games")  
public class GameController  
{  
 @Autowired  
 private GameService gameService;  
  
 @PostMapping()  
 public ResponseEntity AddGame(@RequestBody Game game) throws SQLException  
 {  
 gameService.AddGame(game);  
 return ResponseEntity.ok(HttpStatus.OK);  
 }  
  
 @DeleteMapping("/{id}")  
 public ResponseEntity DeleteGame(@PathVariable Long id) throws SQLException  
 {  
 gameService.DeleteGame(id);  
 return ResponseEntity.ok(HttpStatus.OK);  
 }  
  
 @GetMapping()  
 public List<Game> GetAllGame() throws SQLException  
 {  
 return gameService.GetGames();  
 }  
  
 @GetMapping("/{id}")  
 public Game GetGameById(@PathVariable Long id) throws SQLException  
 {  
 return gameService.GetGameById(id);  
 }  
}

Файл: service/GameService.java

package org.task15.service;  
  
import lombok.AllArgsConstructor;  
import org.springframework.stereotype.Service;  
import org.task15.entity.Game;  
import org.task15.repository.GameRepository;  
  
import java.sql.SQLException;  
import java.util.List;  
  
@Service  
@AllArgsConstructor  
public class GameService  
{  
 private GameRepository gameRepository;  
  
 public void AddGame(Game game)  
 {  
 gameRepository.AddGame(game);  
 }  
  
 public void DeleteGame(Long id)  
 {  
 gameRepository.DeleteGame(id);  
 }  
  
 public List<Game> GetGames()  
 {  
 return gameRepository.GetAllGame();  
 }  
  
 public Game GetGameById(Long id)  
 {  
 return gameRepository.GetGameById(id);  
 }  
}

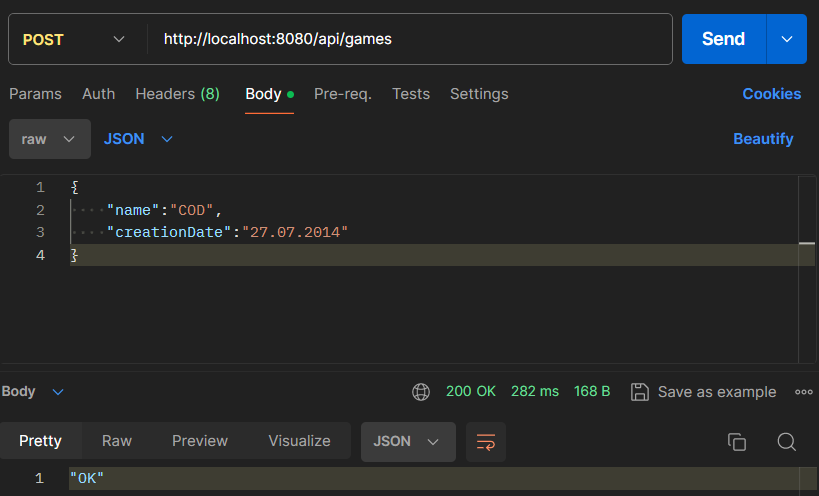
Файл: repository/GameRepository.java

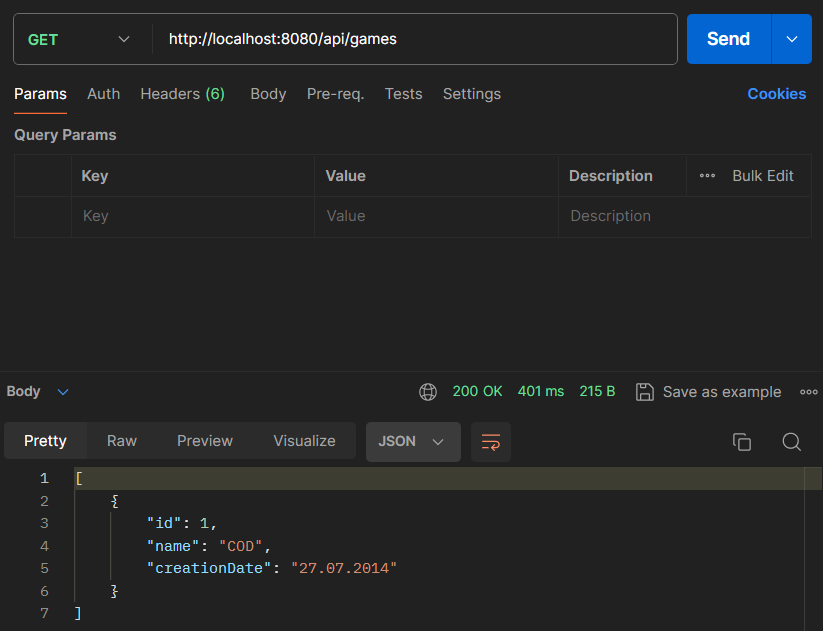
package org.task15.repository;  
  
import jakarta.annotation.PostConstruct;  
import lombok.RequiredArgsConstructor;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.springframework.stereotype.Repository;  
import org.task15.entity.Game;  
  
import java.util.List;  
  
@Repository  
@RequiredArgsConstructor  
public class GameRepository  
{  
 private final SessionFactory sessionFactory;  
 private Session session;  
  
 @PostConstruct  
 void init()  
 {  
 session = sessionFactory.openSession();  
 }  
  
 public void AddGame(Game game)  
 {  
 var t = session.beginTransaction();  
 session.persist(game);  
 t.commit();  
 }  
  
 public void DeleteGame(Long id)  
 {  
 var t = session.beginTransaction();  
 session.remove(session.find(Game.class, id));  
 t.commit();  
 }  
  
 public List<Game> GetAllGame()  
 {  
 var t = session.beginTransaction();  
 var games = session.createQuery("select g from Game g", Game.class).getResultList();  
 t.commit();  
 return games;  
 }  
  
 public Game GetGameById(Long id)  
 {  
 var t = session.beginTransaction();  
 var game = session.find(Game.class, id);  
 t.commit();  
 return game;  
 }  
}

Файл: BeanConfig.java

package org.task15;  
  
import com.zaxxer.hikari.HikariConfig;  
import com.zaxxer.hikari.HikariDataSource;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.orm.hibernate5.HibernateTransactionManager;  
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;  
import org.springframework.transaction.PlatformTransactionManager;  
  
import javax.sql.DataSource;  
import java.util.Properties;  
  
@Configuration  
public class BeanConfig  
{  
 @Bean  
 public HikariDataSource dataSource()  
 {  
 var config = new HikariConfig();  
 config.setJdbcUrl("jdbc:postgresql://localhost:5432/test");  
 config.setUsername("postgres");  
 config.setPassword("password");  
  
 return new HikariDataSource(config);  
 }  
  
 @Bean(name="entityManagerFactory")  
 public LocalSessionFactoryBean factoryBean(DataSource dataSource)  
 {  
 var sessionFactoryBean = new LocalSessionFactoryBean();  
 sessionFactoryBean.setDataSource(dataSource);  
 sessionFactoryBean.setPackagesToScan("org.task15");  
  
 var properties = new Properties();  
 properties.setProperty("hibernate.dialect", "org.hibernate.dialect.PostgreSQLDialect");  
 properties.setProperty("hibernate.hbm2ddl.auto", "create");  
 sessionFactoryBean.setHibernateProperties(properties);  
  
 return sessionFactoryBean;  
 }  
  
 @Bean  
 public PlatformTransactionManager platformTransactionManager(LocalSessionFactoryBean factoryBean)  
 {  
 var transactionManager = new HibernateTransactionManager();  
 transactionManager.setSessionFactory(factoryBean.getObject());  
 return transactionManager;  
 }  
}

1. **Результат работы программы**





# **Практическая работа №16**

1. **Постановка задачи и персональный вариант**

Тема: 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**

Файл: entity/Game.java

package org.task16.entity;  
  
import jakarta.persistence.\*;  
import lombok.Getter;  
import lombok.Setter;  
  
import java.util.List;  
  
@Entity  
@Table(name = "games")  
@Getter  
@Setter  
public class Game  
{  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 @Column(name="name", nullable=false, length=25)  
 private String name;  
 @Column(name="creation\_date", nullable=false, length=25)  
 private String creationDate;  
  
 @OneToMany(cascade = CascadeType.ALL)  
 @JoinColumn(name = "game\_id")  
 private List<Level> levels;  
}

Файл: entity/Level.java

package org.task16.entity;  
  
import jakarta.persistence.\*;  
import lombok.Getter;  
import lombok.Setter;  
  
@Table(name = "levels")  
@Entity  
@Getter  
@Setter  
public class Level  
{  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 @Column(name="complexity", nullable=false, length=25)  
 private String complexity;  
 @Column(name="level\_name", nullable=false, length=25)  
 private String levelName;  
  
 @Column(name="game\_id")  
 public Long gameId;  
}

Файл: controller/GameRepository.java

package org.task16.repository;  
  
import jakarta.annotation.PostConstruct;  
import lombok.RequiredArgsConstructor;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.springframework.stereotype.Repository;  
import org.task16.entity.Game;  
import org.task16.entity.Level;  
  
import java.util.List;  
  
@Repository  
@RequiredArgsConstructor  
public class GameRepository  
{  
 private final SessionFactory sessionFactory;  
 private Session session;  
  
 @PostConstruct  
 void init()  
 {  
 session = sessionFactory.openSession();  
 }  
  
 public void AddGame(Game game)  
 {  
 var t = session.beginTransaction();  
 session.persist(game);  
 t.commit();  
 }  
  
 public void DeleteGame(Long id)  
 {  
 var t = session.beginTransaction();  
 session.remove(session.find(Game.class, id));  
 t.commit();  
 }  
  
 public List<Game> GetAllGame()  
 {  
 var t = session.beginTransaction();  
 var games = session.createQuery("select g from Game g", Game.class).getResultList();  
  
 for (Game g : games)  
 {  
 var levels = session.createQuery("select l from Level l where l.gameId=%s".formatted(g.getId()), Level.class).getResultList();  
 g.setLevels(levels);  
 }  
  
 t.commit();  
 return games;  
 }  
  
 public Game GetGameById(Long id)  
 {  
 var t = session.beginTransaction();  
 var game = session.find(Game.class, id);  
 t.commit();  
 return game;  
 }  
}

Файл: controller/LevelController.java

package org.task16.controller;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.task16.entity.Game;  
import org.task16.entity.Level;  
import org.task16.service.LevelService;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("api/levels")  
public class LevelController  
{  
 @Autowired  
 private LevelService levelService;  
  
 @PostMapping("/{gameId}")  
 public ResponseEntity AddLevel(@RequestBody Level level,@PathVariable Long gameId)  
 {  
 levelService.AddLevel(level, gameId);  
 return ResponseEntity.ok(HttpStatus.OK);  
 }  
  
 @DeleteMapping("/{id}")  
 public ResponseEntity DeleteLevel(@PathVariable Long id)  
 {  
 levelService.DeleteLevel(id);  
 return ResponseEntity.ok(HttpStatus.OK);  
 }  
  
 @GetMapping()  
 public List<Level> GetAllLevel()  
 {  
 return levelService.GetAllLevel();  
 }  
  
 @GetMapping("/{id}")  
 public Level GetLevelById(@PathVariable Long id)  
 {  
 return levelService.GetLevelById(id);  
 }  
}

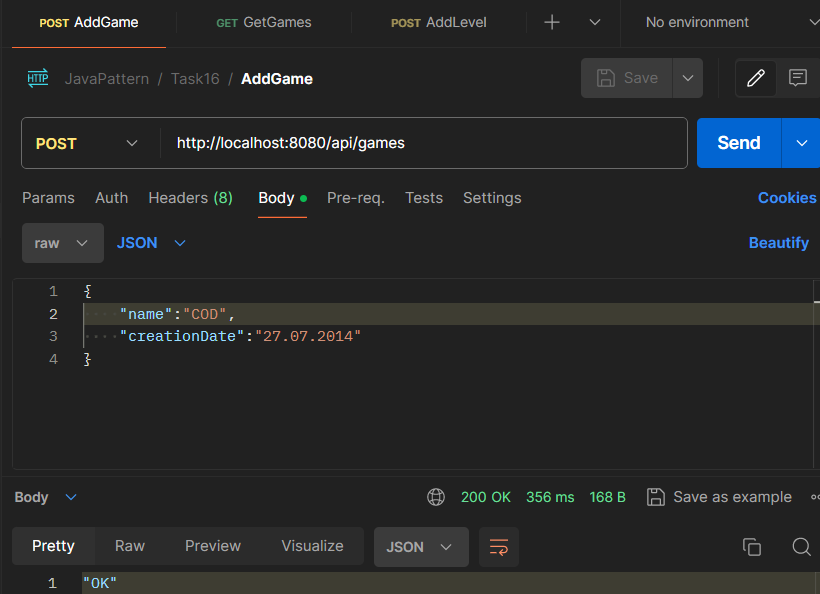
Файл: repository/LevelRepository.java

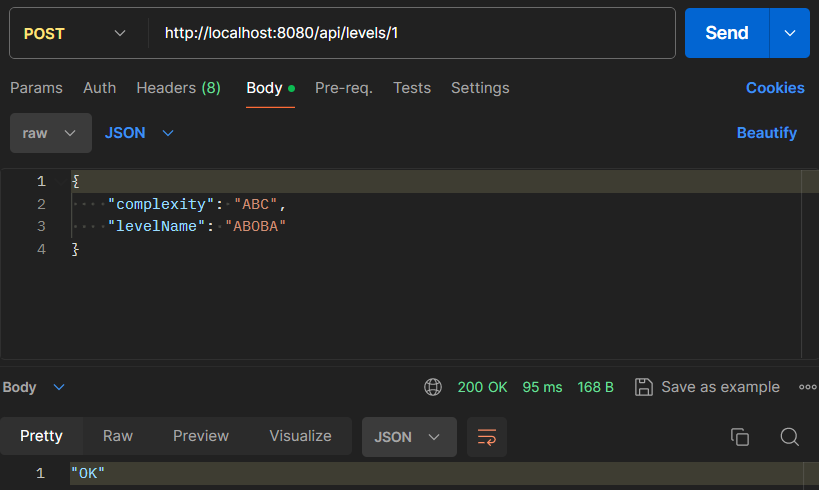
package org.task16.repository;  
  
import jakarta.annotation.PostConstruct;  
import lombok.RequiredArgsConstructor;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.springframework.stereotype.Repository;  
import org.task16.entity.Game;  
import org.task16.entity.Level;  
  
import java.util.List;  
  
@Repository  
@RequiredArgsConstructor  
public class LevelRepository  
{  
 private final SessionFactory sessionFactory;  
 private Session session;  
  
 @PostConstruct  
 void init()  
 {  
 session = sessionFactory.openSession();  
 }  
  
 public void AddLevel(Level level, Long gameId)  
 {  
 var t = session.beginTransaction();  
 var game = session.get(Game.class, gameId);  
 session.persist(level);  
  
 game.getLevels().add(level);  
 session.update(game);  
 t.commit();  
 }  
  
 public void DeleteLevel(Long id)  
 {  
 var t = session.beginTransaction();  
 session.remove(session.find(Level.class, id));  
 t.commit();  
 }  
  
 public List<Level> GetAllLevel()  
 {  
 var t = session.beginTransaction();  
 var levels = session.createQuery("select g from Level g", Level.class).getResultList();  
 t.commit();  
 return levels;  
 }  
  
 public Level GetLevelById(Long id)  
 {  
 var t = session.beginTransaction();  
 var game = session.find(Level.class, id);  
 t.commit();  
 return game;  
 }  
}

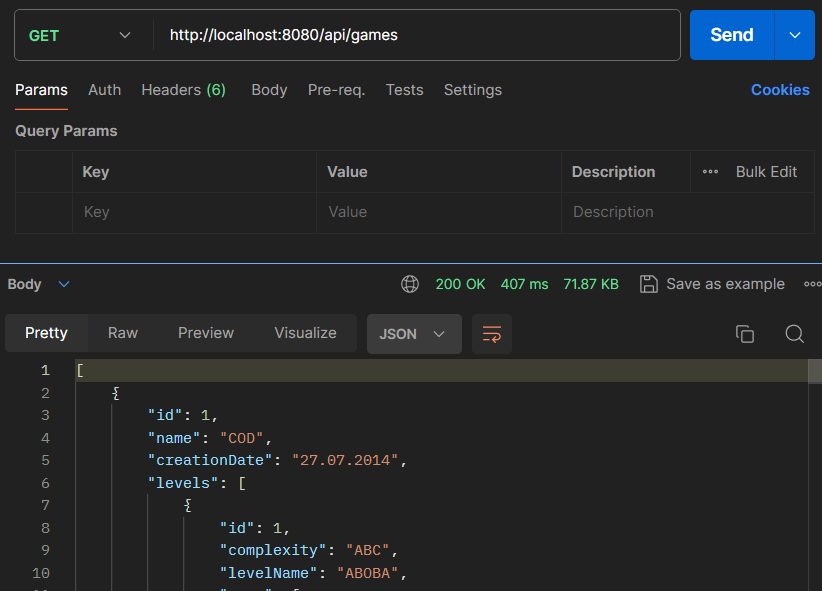
Файл: service/LevelService.java

package org.task16.service;  
  
import lombok.AllArgsConstructor;  
import org.springframework.stereotype.Service;  
import org.task16.entity.Game;  
import org.task16.entity.Level;  
import org.task16.repository.GameRepository;  
import org.task16.repository.LevelRepository;  
  
import java.sql.SQLException;  
import java.util.List;  
  
@Service  
@AllArgsConstructor  
public class LevelService  
{  
 private LevelRepository gameRepository;  
  
 public void AddLevel(Level level, Long gameId)  
 {  
 gameRepository.AddLevel(level, gameId);  
 }  
  
 public void DeleteLevel(Long id)  
 {  
 gameRepository.DeleteLevel(id);  
 }  
  
 public List<Level> GetAllLevel()  
 {  
 return gameRepository.GetAllLevel();  
 }  
  
 public Level GetLevelById(Long id)  
 {  
 return gameRepository.GetLevelById(id);  
 }  
}

1. **Результат работы программы**







# **Практическая работа №17**

1. **Постановка задачи и персональный вариант**

Тема: 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**

Файл: controller/GameController.java

package org.task17.controller;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.task17.entity.Game;  
import org.task17.service.GameService;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("api/games")  
public class GameController  
{  
 @Autowired  
 private GameService gameService;  
  
 @PostMapping()  
 public ResponseEntity AddGame(@RequestBody Game game)  
 {  
 gameService.AddGame(game);  
 return ResponseEntity.ok(HttpStatus.OK);  
 }  
  
 @DeleteMapping("/{id}")  
 public ResponseEntity DeleteGame(@PathVariable Long id)  
 {  
 gameService.DeleteGame(id);  
 return ResponseEntity.ok(HttpStatus.OK);  
 }  
  
 @GetMapping()  
 public List<Game> GetAllGame()  
 {  
 return gameService.GetAllGame();  
 }  
  
 @GetMapping("/{id}")  
 public Game GetGameById(@PathVariable Long id)  
 {  
 return gameService.GetGameById(id);  
 }  
  
 // Filter by name  
 @GetMapping("/name/{pattern}")  
 public List<Game> FilterGamesByName(@PathVariable String pattern)  
 {  
 return gameService.FilterGamesByName(pattern);  
 }  
  
 // Filter by creationDate  
 @GetMapping("/date/{pattern}")  
 public List<Game> FilterGamesByDate(@PathVariable String pattern)  
 {  
 return gameService.FilterGamesByDate(pattern);  
 }  
}

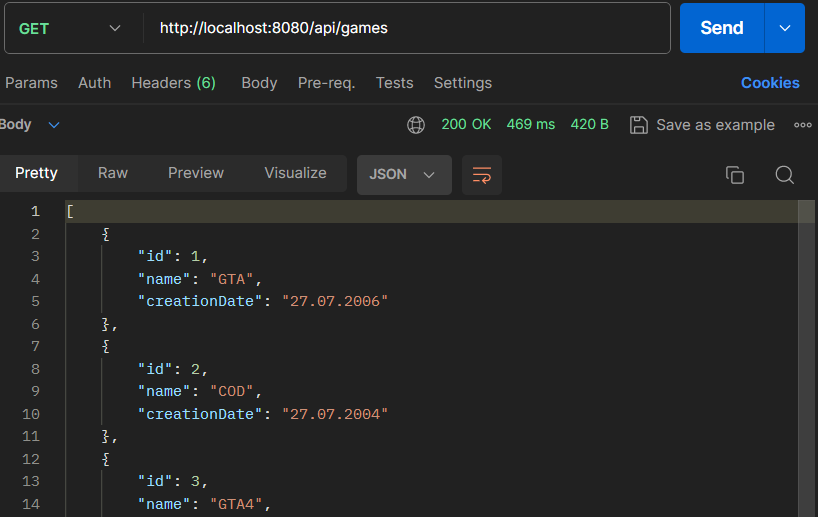
Файл: repository/GameRepository.java

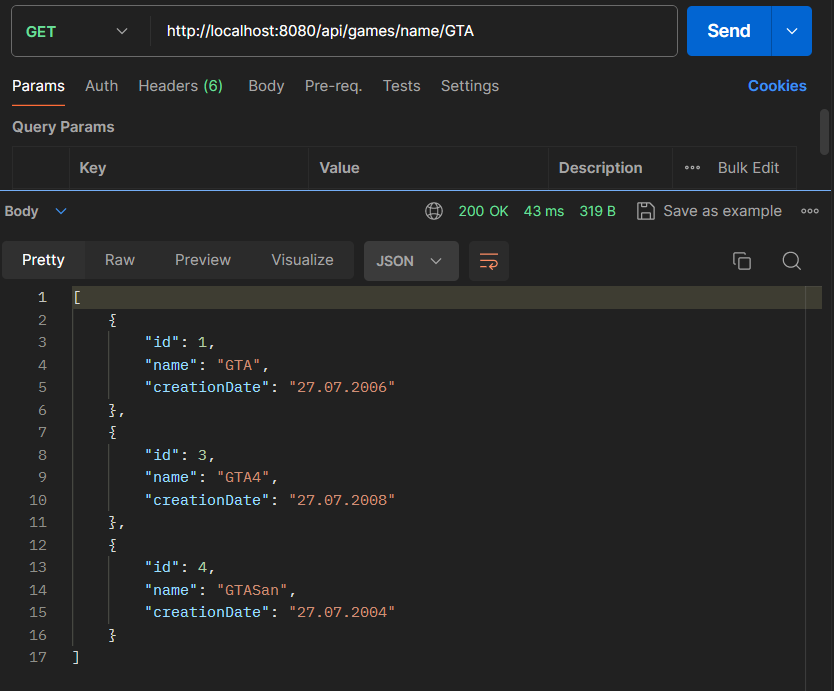
package org.task17.repository;  
  
import jakarta.annotation.PostConstruct;  
import jakarta.persistence.criteria.CriteriaBuilder;  
import jakarta.persistence.criteria.CriteriaQuery;  
import jakarta.persistence.criteria.Root;  
import lombok.RequiredArgsConstructor;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.springframework.stereotype.Repository;  
import org.task17.entity.Game;  
  
import java.util.List;  
  
@Repository  
@RequiredArgsConstructor  
public class GameRepository  
{  
 private final SessionFactory sessionFactory;  
 private Session session;  
  
 @PostConstruct  
 void init()  
 {  
 session = sessionFactory.openSession();  
 }  
  
 public void AddGame(Game game)  
 {  
 var t = session.beginTransaction();  
 session.persist(game);  
 t.commit();  
 }  
  
 public void DeleteGame(Long id)  
 {  
 var t = session.beginTransaction();  
 session.remove(session.find(Game.class, id));  
 t.commit();  
 }  
  
 public List<Game> GetAllGame()  
 {  
 var t = session.beginTransaction();  
 var games = session.createQuery("select g from Game g", Game.class).getResultList();  
 t.commit();  
 return games;  
 }  
  
 public Game GetGameById(Long id)  
 {  
 var t = session.beginTransaction();  
 var game = session.find(Game.class, id);  
 t.commit();  
 return game;  
 }  
  
 // Filter  
 public List<Game> FilterGamesByName(String pattern)  
 {  
 CriteriaBuilder builder = session.getCriteriaBuilder();  
 CriteriaQuery<Game> criteriaQuery = builder.createQuery(Game.class);  
 Root<Game> root = criteriaQuery.from(Game.class);  
  
 var pred = builder.like(root.get("name"), "%"+pattern+"%");  
 var query = criteriaQuery.select(root).where(pred);  
 return session.createQuery(query).getResultList();  
 }  
  
 public List<Game> FilterGamesByDate(String pattern)  
 {  
 CriteriaBuilder builder = session.getCriteriaBuilder();  
 CriteriaQuery<Game> criteriaQuery = builder.createQuery(Game.class);  
 Root<Game> root = criteriaQuery.from(Game.class);  
  
 var pred = builder.like(root.get("creationDate"), "%"+pattern+"%");  
 var query = criteriaQuery.select(root).where(pred);  
 return session.createQuery(query).getResultList();  
 }  
}

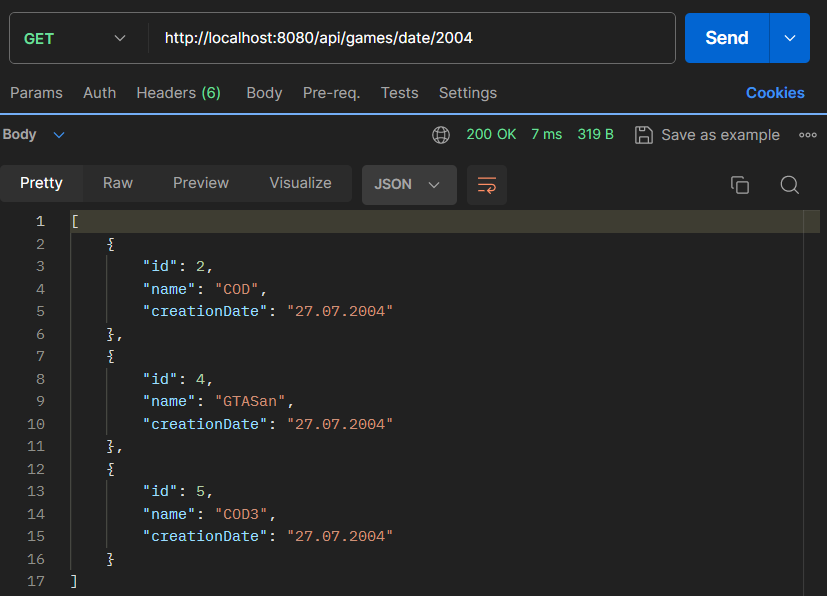
Файл: service/GameService.java

package org.task17.service;  
  
import lombok.AllArgsConstructor;  
import org.springframework.stereotype.Service;  
import org.task17.entity.Game;  
import org.task17.repository.GameRepository;  
  
import java.util.List;  
  
@Service  
@AllArgsConstructor  
public class GameService  
{  
 private GameRepository gameRepository;  
  
 public void AddGame(Game game)  
 {  
 gameRepository.AddGame(game);  
 }  
  
 public void DeleteGame(Long id)  
 {  
 gameRepository.DeleteGame(id);  
 }  
  
 public List<Game> GetAllGame()  
 {  
 return gameRepository.GetAllGame();  
 }  
  
 public Game GetGameById(Long id)  
 {  
 return gameRepository.GetGameById(id);  
 }  
  
 // Filter  
 public List<Game> FilterGamesByName(String pattern)  
 {  
 return gameRepository.FilterGamesByName(pattern);  
 }  
 public List<Game> FilterGamesByDate(String pattern)  
 {  
 return gameRepository.FilterGamesByDate(pattern);  
 }  
}

1. **Результат работы программы**







# **Практическая работа №18**

1. **Постановка задачи и персональный вариант**

Тема: 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**

Файл: repository/GameRepository.java

package org.task18.repository;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
import org.task18.entity.Game;

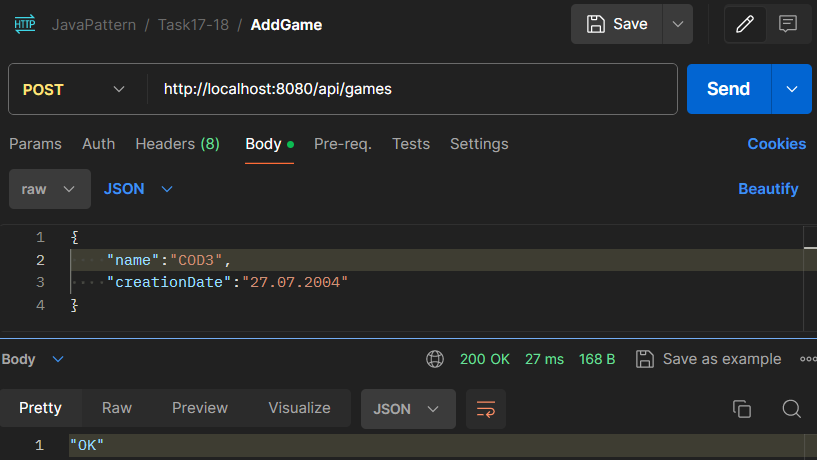
import java.util.List;

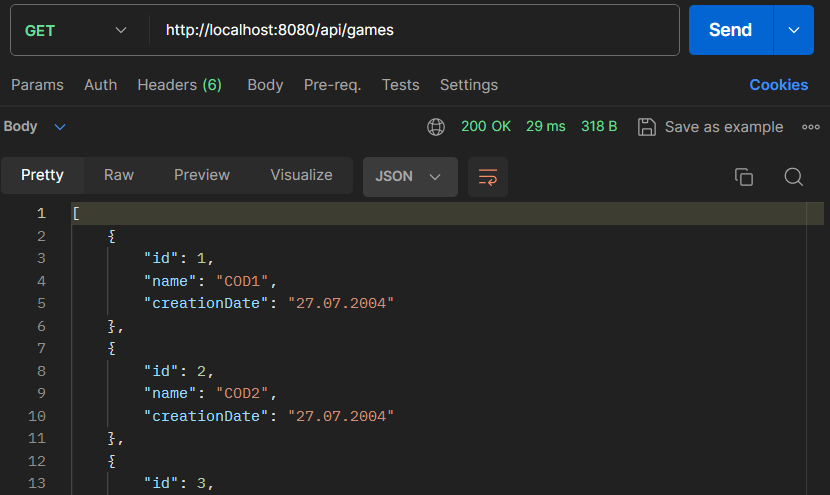
@Repository  
public interface GameRepository extends JpaRepository<Game, Long>  
{  
 List<Game> findAllByNameContains(String name);  
 List<Game> findAllByCreationDateContains(String date);  
}

Файл: application.properties

spring.application.name=Task18  
spring.datasource.driver-class-name=org.postgresql.Driver  
spring.jpa.show-sql=true  
spring.jpa.generate-ddl=true  
spring.datasource.url=jdbc:postgresql://localhost:5432/test  
spring.datasource.username=postgres  
spring.datasource.password=password

1. **Результат работы программы**





# **Практическая работа №19**

1. **Постановка задачи и персональный вариант**

Тема: Знакомство с логированием с использованием Logback в Spring.

Постановка задачи: Создать файл logback.xml, добавить логирование во все методы классовсервисов.

1. **Код**

Файл: service/GameService.java

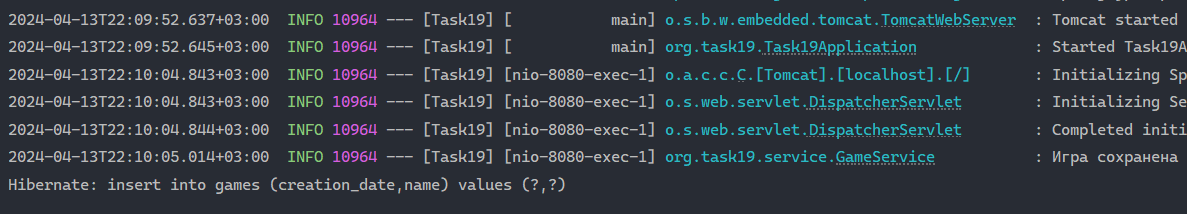
package org.task19.service;  
  
import lombok.AllArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.stereotype.Service;  
import org.task19.entity.Game;  
import org.task19.repository.GameRepository;  
  
import java.util.List;  
  
@Service  
@AllArgsConstructor  
@Slf4j  
public class GameService  
{  
 private GameRepository gameRepository;  
  
 public void AddGame(Game game)  
 {  
 log.info("Игра сохранена");  
 gameRepository.save(game);  
 }  
  
 public void DeleteGame(Long id)  
 {  
 gameRepository.deleteById(id);  
 }  
  
 public List<Game> GetAllGame()  
 {  
 log.info("Получили все игры");  
 return gameRepository.findAll();  
 }  
  
 public Game GetGameById(Long id)  
 {  
 return gameRepository.findById(id).get();  
 }  
  
 // Filter  
 public List<Game> FilterGamesByName(String pattern)  
 {  
 return gameRepository.findAllByNameContains(pattern);  
 }  
 public List<Game> FilterGamesByDate(String pattern)  
 {  
 return gameRepository.findAllByCreationDateContains(pattern);  
 }  
}

Файл: logback.xml

<?xml version="1.0" encoding="UTF-8"?>  
<configuration>  
  
 <property name="LOGS" value="./logs" />  
  
 <appender name="Console"  
 class="ch.qos.logback.core.ConsoleAppender">  
 <layout class="ch.qos.logback.classic.PatternLayout">  
 <Pattern>  
 %black(%d{ISO8601}) %highlight(%-5level) [%blue(%t)] %yellow(%C{1.}): %msg%n%throwable  
 </Pattern>  
 </layout>  
 </appender>  
  
 <appender name="RollingFile"  
 class="ch.qos.logback.core.rolling.RollingFileAppender">  
 <file>${LOGS}/spring-boot-logger.log</file>  
 <encoder  
 class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">  
 <Pattern>%d %p %C{1.} [%t] %m%n</Pattern>  
 </encoder>  
  
 <rollingPolicy  
 class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
 <!-- rollover daily and when the file reaches 10 MegaBytes -->  
 <fileNamePattern>${LOGS}/archived/spring-boot-logger-%d{yyyy-MM-dd}.%i.log  
 </fileNamePattern>  
 <timeBasedFileNamingAndTriggeringPolicy  
 class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  
 <maxFileSize>10MB</maxFileSize>  
 </timeBasedFileNamingAndTriggeringPolicy>  
 </rollingPolicy>  
 </appender>  
  
 <!-- LOG everything at INFO level -->  
 <root level="info">  
 <appender-ref ref="RollingFile" />  
 <appender-ref ref="Console" />  
 </root>

<!-- LOG "com.baeldung\*" at TRACE level -->  
 <logger name="com.baeldung" level="trace" additivity="false">  
 <appender-ref ref="RollingFile" />  
 <appender-ref ref="Console" />  
 </logger>  
  
</configuration>

1. **Результат работы программы**



# **Практическая работа №20**

1. **Постановка задачи и персональный вариант**

Тема: Использование Spring AOP. Pointcut, JoinPoint. Advice

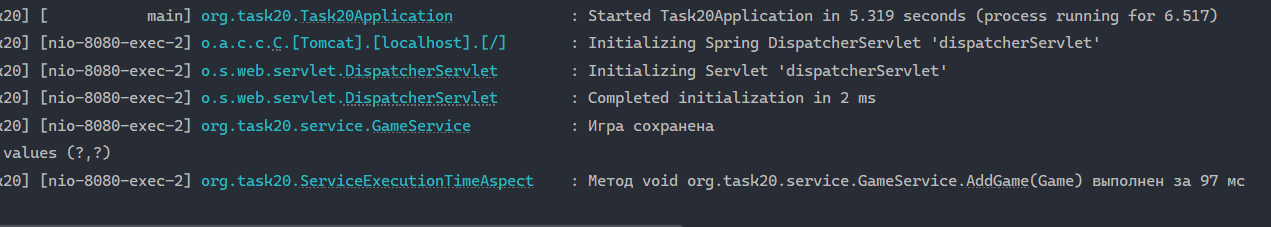
Постановка задачи: Для приложения из предыдущего задания добавить логирование времени выполнения каждого метода сервиса с использованием Spring AOP.

1. **Код**

Файл: ServiceExecutionTimeAspect.java

package org.task20;  
  
import lombok.extern.slf4j.Slf4j;  
import org.aspectj.lang.JoinPoint;  
import org.aspectj.lang.annotation.After;  
import org.aspectj.lang.annotation.Aspect;  
import org.aspectj.lang.annotation.Before;  
import org.springframework.stereotype.Component;  
  
@Aspect  
@Component  
@Slf4j  
public class ServiceExecutionTimeAspect {  
 private long startTime;  
  
 @Before("execution(\* org.task20.service.GameService.\*(..))")  
 public void beforeMethodExecution() {  
 startTime = System.currentTimeMillis();  
 }  
  
 @After("execution(\* org.task20.service.GameService.\*(..))")  
 public void afterMethodExecution(JoinPoint joinPoint) {  
 long executionTime = System.currentTimeMillis() - startTime;  
 log.info("Метод " + joinPoint.getSignature() + " выполнен за " + executionTime + " мс");  
 }  
}

1. **Результат работы программы**



# **Практическая работа №21**

1. **Постановка задачи и персональный вариант**

Тема: Проксирование. Аннотация Transactional. Аннотация Async.

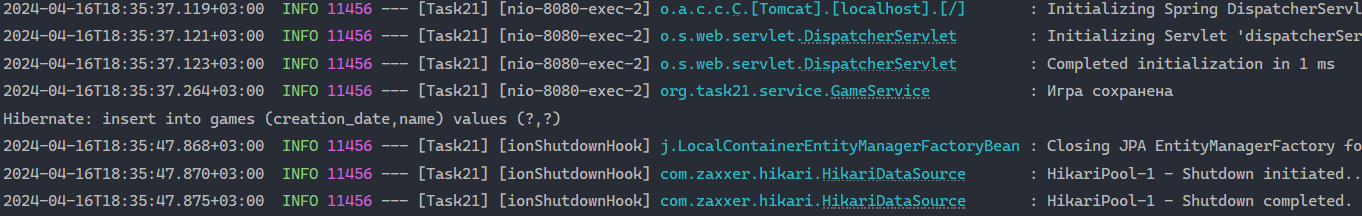
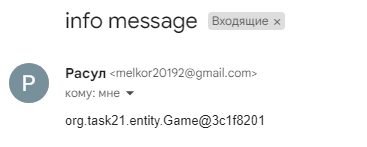
Постановка задачи: Для приложения из предыдущего задания пометить все классы сервисов, в которых происходит взаимодействие с базой данных, как Transactional. Добавить отправку информации о сохранении каждого объекта по электронной почте, создав отдельный класс EmailService с асинхронными методами отправки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

1. **Код**

Файл: EmailSender.java

package org.task21;  
  
import lombok.AllArgsConstructor;  
import org.springframework.mail.SimpleMailMessage;  
import org.springframework.mail.javamail.JavaMailSender;  
import org.springframework.scheduling.annotation.Async;  
import org.springframework.stereotype.Service;  
import org.task21.entity.Game;  
  
@Service  
@AllArgsConstructor  
public class EmailSender  
{  
 private final JavaMailSender javaMailSender;  
  
 @Async  
 public void sendEmail(Game game) {  
 SimpleMailMessage message = new SimpleMailMessage();  
  
 message.setFrom("melkor20192@gmail.com");  
 message.setTo("melkor20192@gmail.com ");  
 message.setSubject("Info message");  
 message.setText(game.toString());  
  
 javaMailSender.send(message);  
 }  
}

1. **Результат работы программы**

# **Практическая работа №22**

1. **Постановка задачи и персональный вариант**

Тема: 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**

Файл: ScheduledService.java

package org.task22.service;  
  
import lombok.AllArgsConstructor;

import org.springframework.scheduling.annotation.EnableScheduling;

import org.springframework.scheduling.annotation.Scheduled;

import org.springframework.stereotype.Service;  
  
import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.util.Objects;  
  
@Service  
@EnableScheduling  
@AllArgsConstructor  
public class ScheduledService

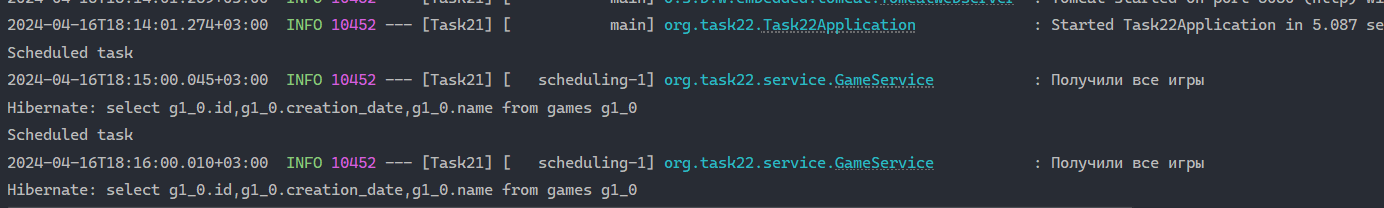
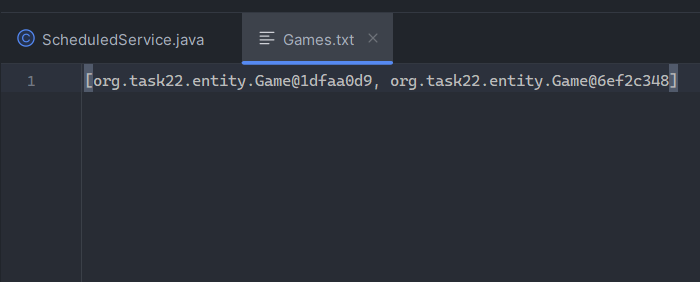
{  
 private final GameService service;  
  
 @Scheduled(cron = "0 \* \* \* \* \*")  
 public void task() throws IOException

{  
 String path = "D:\\Programming\\JavaPatterns\\Task22\\src\\main\\resources\\entities";  
 System.out.println("Scheduled task");

for (File myFile : Objects.requireNonNull(new File(path).listFiles()))

{  
 if (myFile.isFile()) myFile.delete();  
 }  
  
 File file = new File(path +"/Games.txt");  
 file.createNewFile();  
 FileWriter fileWriter = new FileWriter(file);  
 fileWriter.write(service.GetAllGame().toString());  
 fileWriter.close();  
 }  
}

1. **Результат работы программы**

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Стелтинг С., Маасен О. Применение шаблонов Java. Библиотека профессионала.: Пер. с англ. — М.: Издательский дом "Вильяме", 2002. — 576 с.: ил. — Парал. тит. англ.
2. Functional Interfaces in Java: Fundamentals and Examples 1st ed. Edition, Kindle Edition [Электронный ресурс]. URL: https://www.amazon.com/Functional-Interfaces-Java-Fundamentals-Examplesebook/dp/B07NRHQSCW (дата обращения: 29.01.21). Заголовок с экрана.
3. Hibernate Search 6.0.0.Final: Reference Documentation [Электронный ресурс].URL:https://docs.jboss.org/hibernate/stable/search/reference/enUS/html\_single/ (дата обращения: 29.01.21). Заголовок с экрана.
4. Паттерны проектирования на Java. Каталог Java-примеров. [Электронный ресурс]. URL: https://refactoring.guru/ru/design-patterns/java (дата обращения: 29.01.21). Заголовок с экрана.
5. Руководство по Spring [Электронный ресурс]. URL: https://proselyte.net/tutorials/spring-tutorial-full-version/ (дата обращения: 29.01.21). Заголовок с экрана.
6. The Reactive Manifesto [Электронный ресурс]. URL: https://www.reactivemanifesto.org/ (дата обращения: 29.01.21). Заголовок с экрана.
7. Spring Framework Documentation [Электронный ресурс]. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/web.html (дата обращения: 29.01.21). Заголовок с экрана.
8. Hibernate Search 6.0.0. Final: Reference Documentation [Электронный ресурс].URL:https://docs.jboss.org/hibernate/stable/search/reference/enUS/html\_single/ (дата обращения: 29.01.21). Заголовок с экрана.