|  |
| --- |
|  |
| 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-20-22** | ***Исаков Л.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**

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.

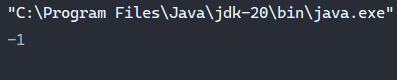
1. **Код**

Папка: <https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task1>

Файл: Main.java

package Task1;  
  
import java.util.Comparator;  
  
public class Main  
{  
 public static void main(String[] args)  
 {  
 Comparator<Object[]> comparator = (arr1, arr2) -> arr1.length - arr2.length;  
 var newArr1 = new Integer[] {1, 2};  
 var newArr2 = new Integer[] {2, 3, 4};  
 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**

Папка: <https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task2>

Файл: Main.java

package Task2;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.Comparator;  
  
public class Main  
{  
 public static void main(String[] args)  
 {  
 var list = new ArrayList<Human>();  
 list.add(new Human(18, "Alex", "Alexandrov", LocalDate.of(2005, 7, 1), 60));  
 list.add(new Human(19, "Sveta", "Oblakova", LocalDate.of(2004, 7, 1), 55));  
 list.add(new Human(20, "Lesha", "Terekov", LocalDate.of(2003, 7, 1), 50));  
 list.add(new Human(21, "Timur", "Ivanov", LocalDate.of(2002, 7, 1), 45));  
 list.add(new Human(22, "Denis", "Petrov", LocalDate.of(2001, 7, 1), 40));  
 list.add(new Human(23, "Nastya", "Sidorova", LocalDate.of(2000, 7, 1), 35));  
  
 var stream = list.stream();  
 var av = stream.filter(human -> human.age > 20)  
 .sorted(new HumanComparator())  
 .peek(human -> human.age += 3)  
 .mapToInt(Human::getAge).average();  
  
 System.out.println(av);  
 }  
}  
  
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**

Папка: <https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task3_1>

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)  
 {  
 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();  
 }  
}

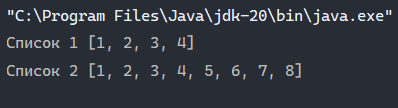
Папка: <https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task3_2>

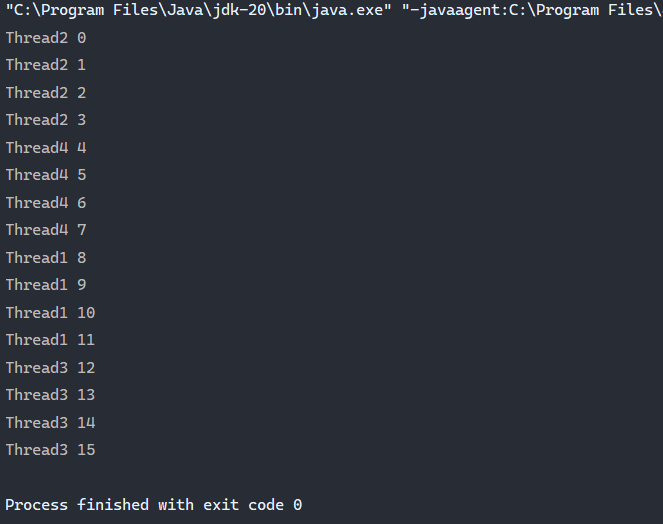
Файл: Main.java

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.

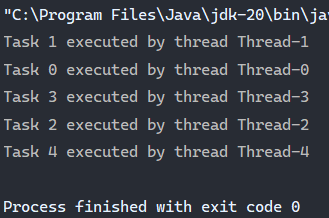
1. **Код**

Папка: <https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task4>

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)  
 {  
 CustomExecutorService executorService = new CustomExecutorService(5);  
  
 for (int i = 0; i < 10; 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.

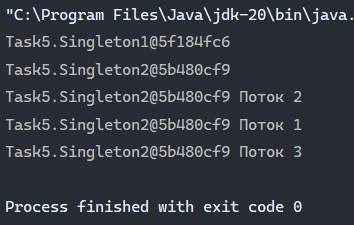
1. **Код**

Папка: <https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task5>

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**

Папка: <https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task6>

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();  
 }  
}

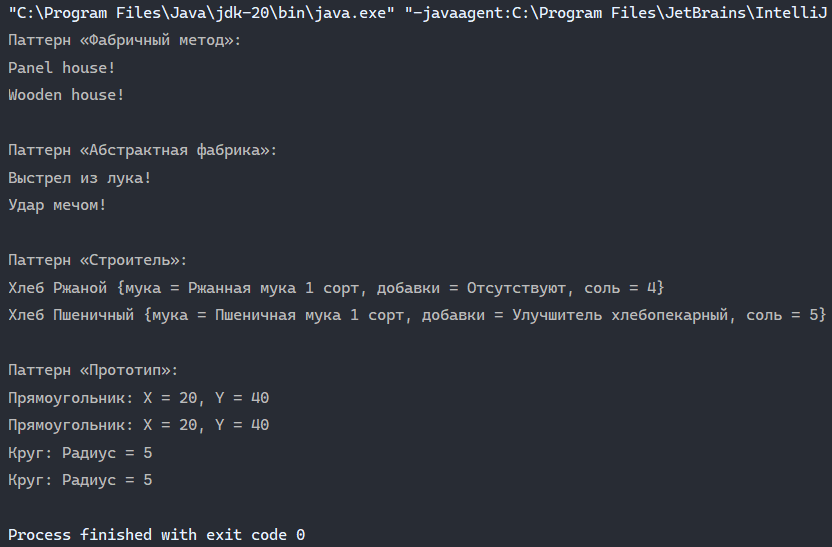
Файл: Builder.java

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

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**

Папка: <https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task7>

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();  
 }  
}

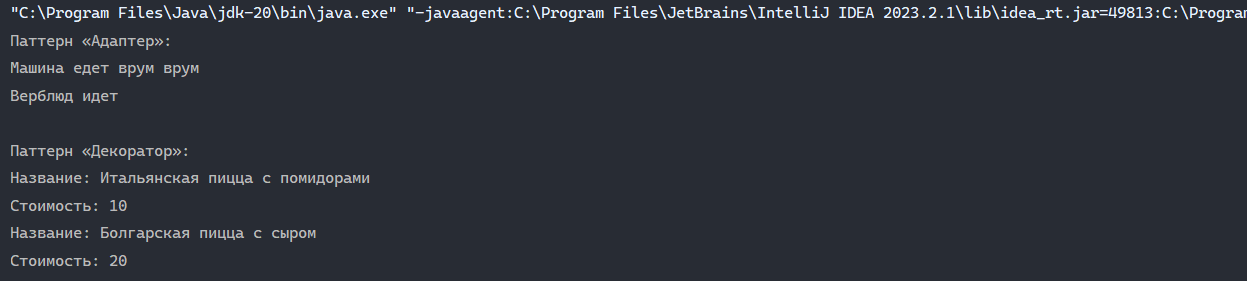
Файл: Adapter.java

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**

Папка: <https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task8>

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();  
 }  
}

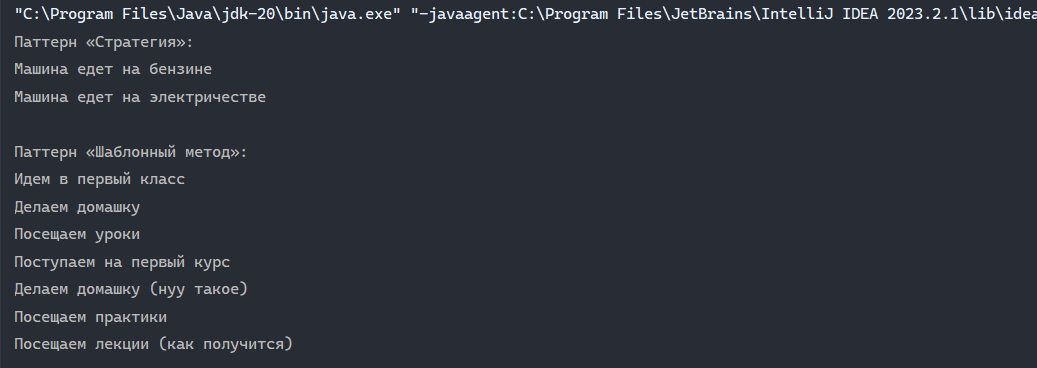
Файл: Shablon.java

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**

Папка: <https://github.com/Peygy/JavaPatterns/tree/main/Task9_Gradle>

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

package ru.mirea;  
  
public class Main  
{  
 public static void main(String[] args)  
 {  
 System.out.println("HELLOOOOOOOOOOOOO!!!!!!!!!!!! 9 Taskaaaa");  
 }  
}

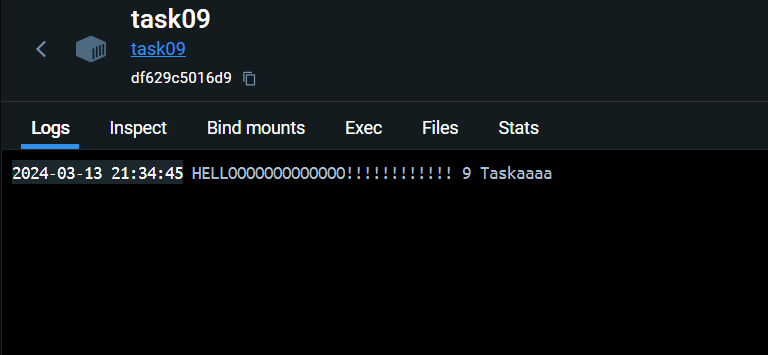
Файл: build.gradle

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**

Папка: <https://github.com/Peygy/JavaPatterns/tree/main/Task10/src/main/java/org/project/task10>

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");  
 }  
}

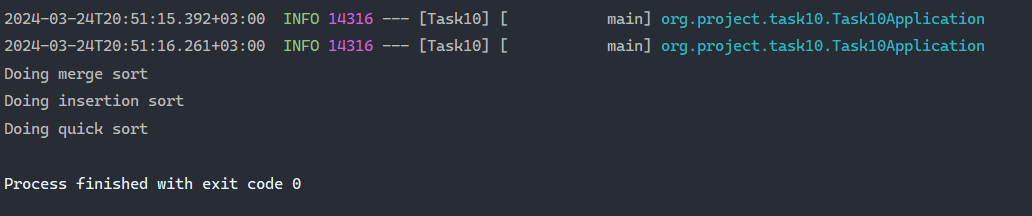
Файл: components/InsertionSort.java

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**

Папка: <https://github.com/Peygy/JavaPatterns/tree/main/Task11>

Файл: src/main/java/com/example/Task11Application.java

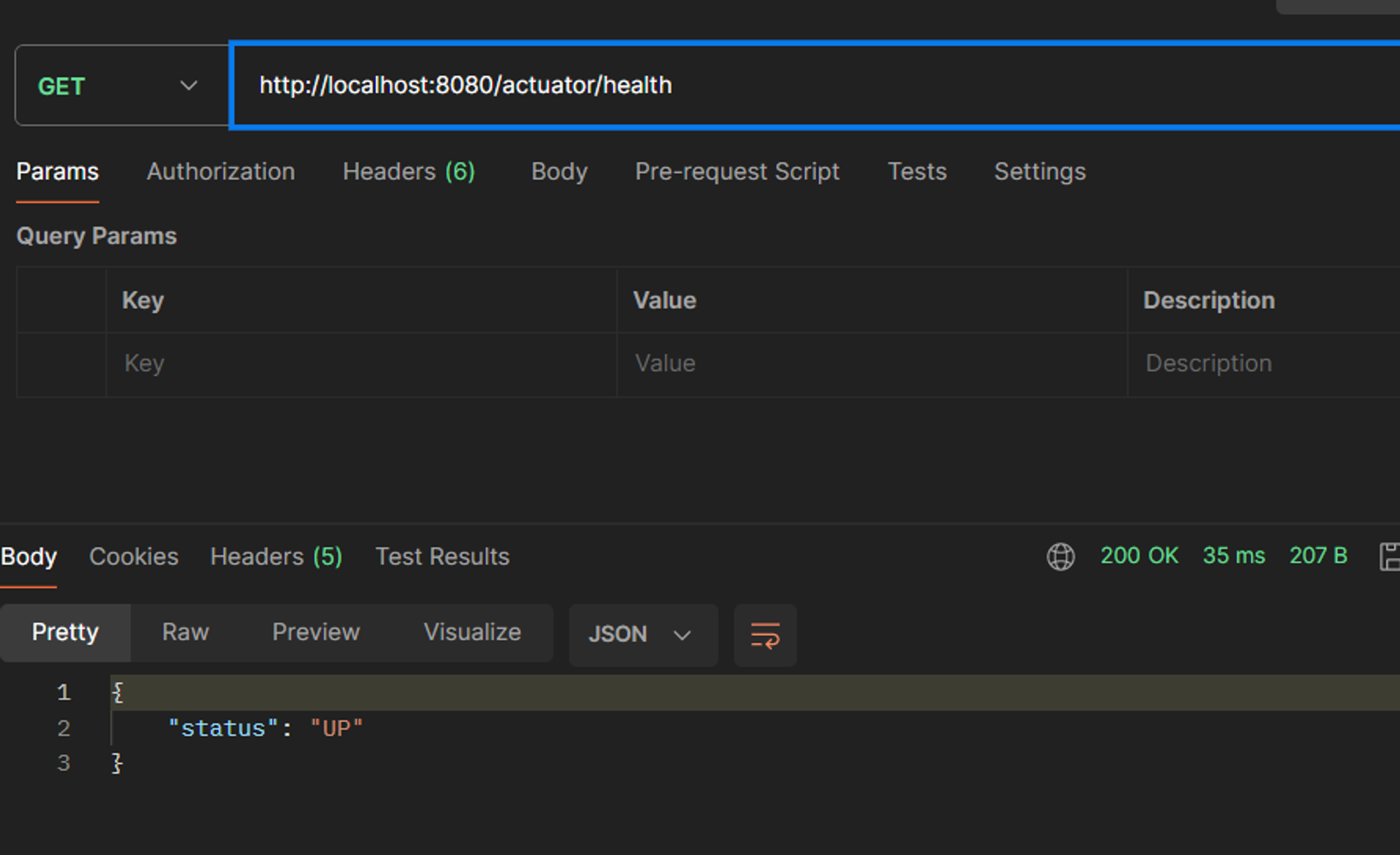
package com.example;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class Task11Application  
{  
 public static void main(String[] args)  
 {  
 SpringApplication.run(Task11Application.class,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

Файл: src/main/java/com/example/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**

Папка: <https://github.com/Peygy/JavaPatterns/tree/main/Task12>

Файл: src/main/java/org/project/task12/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

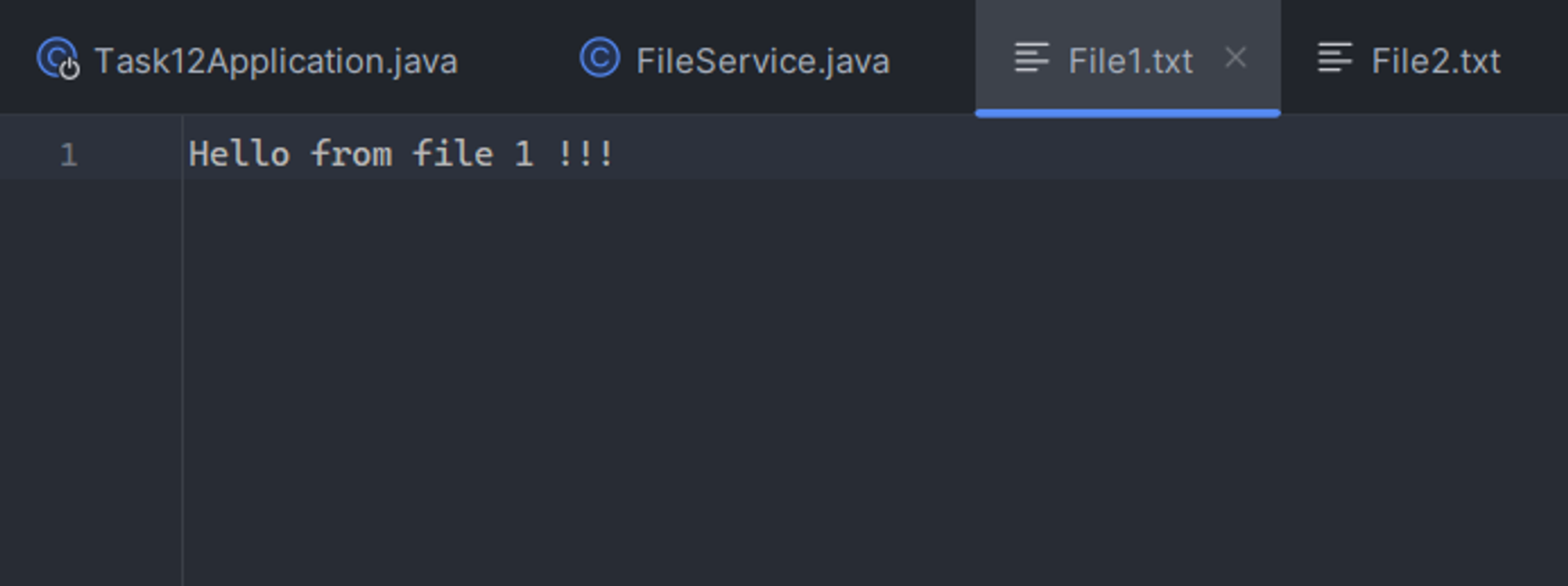
{  
 public static void main(String[] args)  
 {  
 ApplicationContext context = SpringApplication.run(Task12Application.class, args);  
 new FileService();  
 SpringApplication.exit(context);  
 }  
}

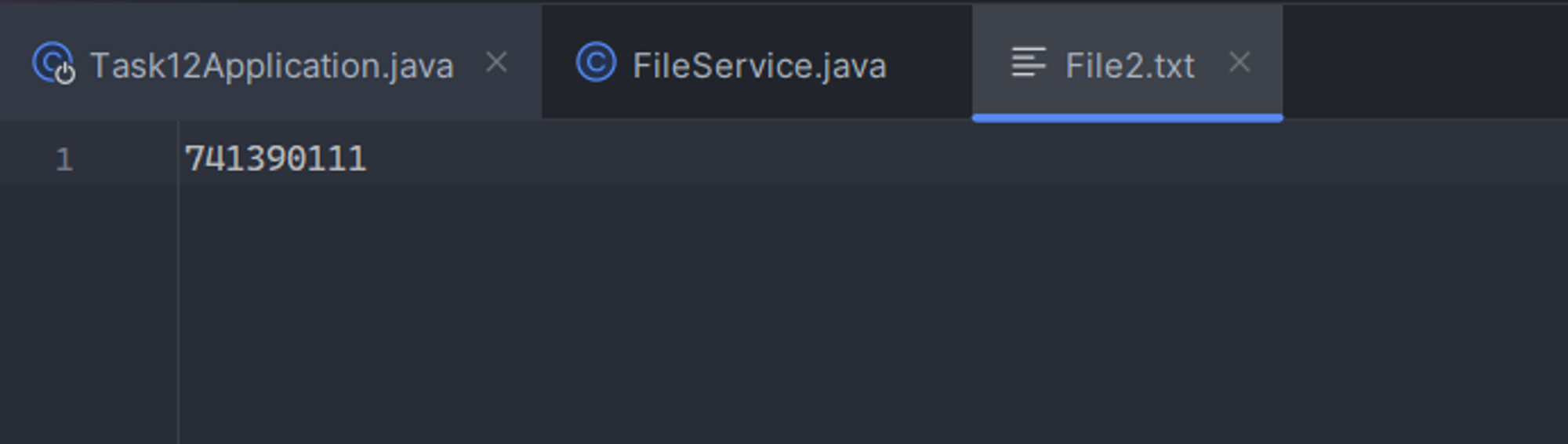
Файл: src/main/java/org/project/task12/FileService.java

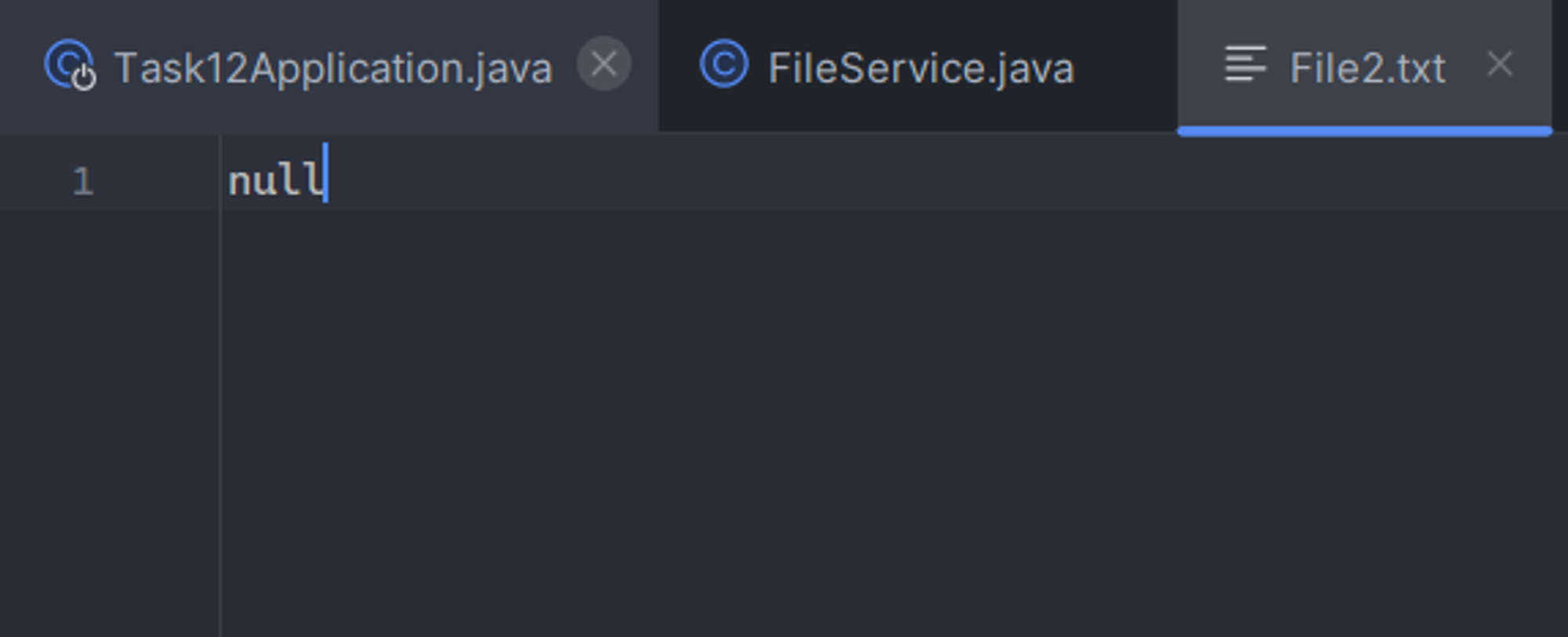
package org.project.task12;  
  
import jakarta.annotation.PostConstruct;  
import jakarta.annotation.PreDestroy;  
import org.springframework.stereotype.Service;  
import java.io.\*;  
  
@Service  
public class FileService  
{  
 String file1 = "D:\\Programming\\JavaPatterns\\Task12\\File1.txt";  
 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: 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**

Папка: <https://github.com/Peygy/JavaPatterns/tree/main/Task11>

Файл: src/main/java/com/example/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: src/main/java/com/example/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**

