|  |
| --- |
|  |
| 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-10-20** | ***Гегия Д.*** |
| Принял преподаватель кафедры ИиППО | ***Волков М.Ю.*** |

|  |  |  |
| --- | --- | --- |
| Практические работы выполнены | « » 2022г. |  |
| «Зачтено» | « » 2022г. |  |

Москва 2022

ОГЛАВЛЕНИЕ

[Практическая работа №1 24](#_Toc102577740)

[Практическая работа №2 25](#_Toc102577741)

[Практическая работа №3 29](#_Toc102577742)

[Практическая работа №4 33](#_Toc102577743)

[Практическая работа №5 35](#_Toc102577754)

[Практическая работа № 6 37](#_Toc102577768)

[Практическая работа №7 46](#_Toc102577807)

[Практическая работа №8 50](#_Toc102577827)

[Практическая работа №9 56](#_Toc102577850)

[Практическая работа №10 58](#_Toc102577863)

[Практическая работа №11 60](#_Toc102577879)

[Практическая работа №12 62](#_Toc102577898)

[Практическая работа №13 65](#_Toc102577911)

[Практическая работа №14 67](#_Toc102577928)

[Практическая работа №15 73](#_Toc102577948)

[Практическая работа №16 81](#_Toc102577966)

[Практическая работа № 17 82](#_Toc102577979)

[Практическая работа №18 84](#_Toc102577992)

[Практическая работа №19 85](#_Toc102578003)

[Практическая работа №20 86](#_Toc102578013)

[Практическая работа № 21 88](#_Toc102578023)

[Практическая работа №22 89](#_Toc102578033)

[Практическая работа № 23 91](#_Toc102578045)

[Практическая работа №24 95](#_Toc102578058)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 100](#_Toc102578071)

**Практическая работа №1**

1. Постановка задачи и персональный вариант

Тема: Знакомство со встроенными функциональными интерфейсами Java.

Возможности Java 8. Лямбда-выражения. Области действия, замыкания.

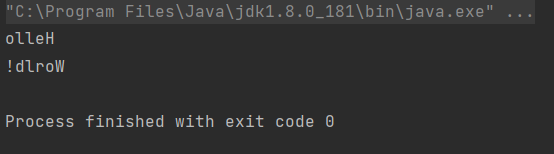
Предикаты. Функции. Компараторы.

Постановка задачи: Имплементировать интерфейс Function, получающий на вход массив строк и возвращающий массив отзеркаленных строк.

1. Код и пояснение к коду

import java.util.function.Function;  
  
public class Main {  
 public static void main(String[] args) {  
 String[] arr = new String[2];  
 arr[0] = "Hello";  
 arr[1] = "World!";  
 Function<String[], String[]> reverseString = input -> {  
 String[] output = new String[input.length];  
 for(int a = 0; a<input.length; a++){  
 output[a] = new StringBuilder(input[a]).reverse().toString();  
 }  
 return output;  
 };  
 arr = reverseString.apply(arr);  
 for (String str:arr  
 ) {  
 System.out.println(str);  
 }  
 }  
}

1. Результат работы программы



1. Выводы по проделанной работе

В результате выполнения практической работы мы познакомились с лямбда-функциями и научились применять их на практике.

**Практическая работа №2**

1. Постановка задачи и персональный вариант

Тема: Работа со Stream API в Java 8.

Постановка задачи: В ходе выполнения практической работы должно быть реализовано:

1) класс Human (int age, String firstName, String

lastName, LocalDate birthDate, int weight);

2) приложение, которое создает список из объектов класса Human, а

затем производит действия в соответствии с вариантом индивидуального

задания (список после каждого этапа должен выводиться в консоль).

Все действия должны производиться только с использованием Stream

API.

Сортировка по весу в обратном порядке, фильтрация по фамилии не

Иванов, сортировка по возрасту, произведение всех возрастов.

1. Код и пояснение к коду

Human.java

import java.time.LocalDate;  
  
public class Human {  
 private int age;  
 private String firstName;  
 private String lastName;  
 private LocalDate birthDate;  
 private int weight;  
  
 public Human(int age, String firstName, String lastName, LocalDate birthDate, int weight) {  
 this.age = age;  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.birthDate = birthDate;  
 this.weight = weight;  
 }  
  
 public int getAge() {  
 return age;  
 }  
  
 public void setAge(int age) {  
 this.age = age;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
  
 public LocalDate getBirthDate() {  
 return birthDate;  
 }  
  
 public void setBirthDate(LocalDate birthDate) {  
 this.birthDate = birthDate;  
 }  
  
 public int getWeight() {  
 return weight;  
 }  
  
 public void setWeight(int weight) {  
 this.weight = weight;  
 }  
  
 @Override  
 public String toString() {  
 return "Human{" +  
 "age=" + age +  
 ", firstName='" + firstName + '\'' +  
 ", lastName='" + lastName + '\'' +  
 ", birthDate=" + birthDate +  
 ", weight=" + weight +  
 '}';  
 }  
}

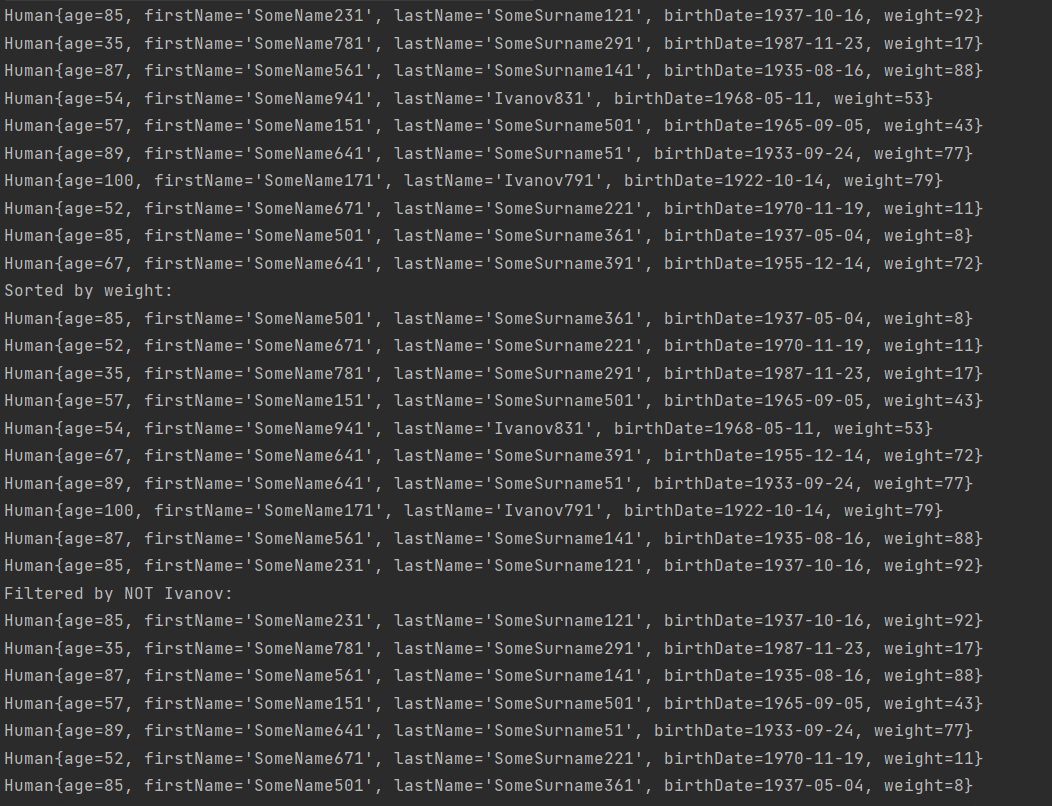
App.java

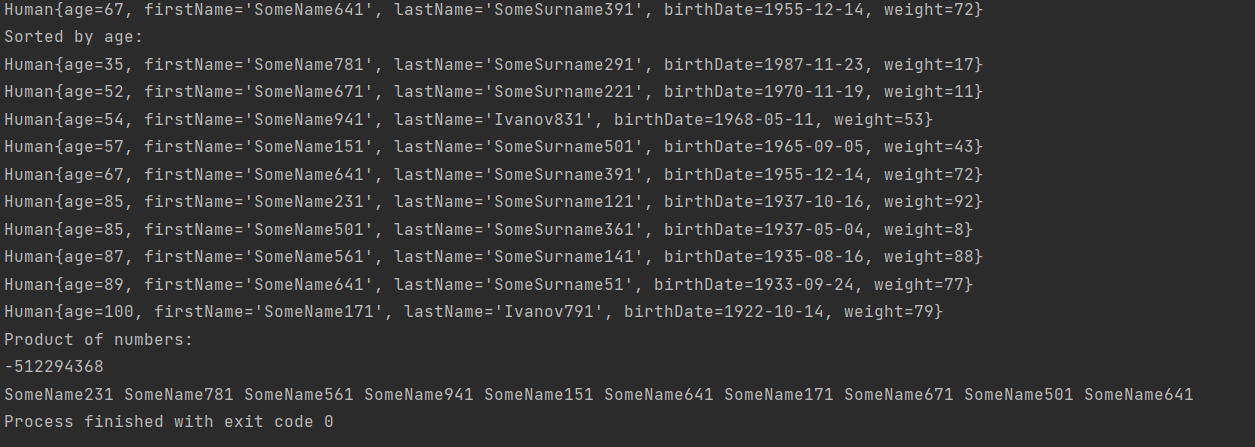
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.Random;  
  
public class App {  
 ArrayList <Human> humans = new ArrayList<>();  
 void addRandomHuman(){  
 int randAge = (int)((Math.*random*() \* ((100 - 1) + 1)) + 1);  
 Random random = new Random();  
 boolean flag = random.nextBoolean();  
 String surname;  
 if(flag){  
 surname = "Ivanov";  
 }  
 else {  
 surname = "SomeSurname";  
 }  
 LocalDate randDate = LocalDate.*of*(2022-randAge,(int)(Math.*random*() \* ((12 - 1) + 1)) + 1, (int)(Math.*random*() \* ((28 - 1) + 1)) + 1);  
 Human human = new Human(randAge, "SomeName" + (int)(Math.*random*() \* ((100 - 1) + 1)) + 1, surname + (int)(Math.*random*() \* ((100 - 1) + 1)) + 1, randDate, (int)(Math.*random*() \* ((100 - 1) + 1)) + 1);  
 humans.add(human);  
 }  
 void outList(){  
 for (Human human:humans  
 ) {  
 System.*out*.println(human.toString());  
 }  
 }  
  
 public ArrayList<Human> getHumans() {  
 return humans;  
 }  
}

Main.java

import java.util.Arrays;  
import java.util.Comparator;  
import java.util.stream.Stream;  
  
public class Main {  
 public static void main(String[] args) {  
 App app = new App();  
 for(int a = 0; a<10; a++){  
 app.addRandomHuman();  
 }  
 app.outList();  
 System.*out*.println("Sorted by weight:");  
 Stream<Human> stream = app.getHumans().stream();  
 stream.sorted(Comparator.*comparingInt*(Human::getWeight)).forEach(System.*out*::println);  
 System.*out*.println("Filtered by NOT Ivanov:");  
 Stream<Human> stream2 = app.getHumans().stream();  
 stream2.filter(human -> !human.getLastName().contains("Ivanov")).forEach(System.*out*::println);  
 Stream<Human> stream3 = app.getHumans().stream();  
 System.*out*.println("Sorted by age:");  
 stream3.sorted(Comparator.*comparingInt*(Human::getAge)).forEach(System.*out*::println);  
 Stream<Human> stream4 = app.getHumans().stream();  
 System.*out*.println("Product of numbers:");  
 int product = stream4.mapToInt(Human::getAge).reduce(1, (a, b) -> a \* b);  
 System.*out*.println(product);  
  
 Stream<Human> stream5 = app.getHumans().stream();  
 stream5.flatMap((input) -> Arrays.*asList*(input.getFirstName() + " ").stream()).forEach(System.*out*::print);  
 }  
}

1. Результат работы программы





1. Выводы о проделанной работе

В результате выполнения данной практической работе мы познакомились с Stream API и научились применять его на практике.

**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 в соответствии с вариантом индивидуального задания

Map с использованием Lock, Set с использованием ключевого слова

Synchronized

1. Код и пояснение к коду

CustomMap.java

import java.util.Collection;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.Set;  
import java.util.concurrent.locks.Lock;  
import java.util.concurrent.locks.ReentrantLock;  
  
public class CustomMap implements Map{  
 Map map;  
 public CustomMap(){  
 map = new HashMap();  
 }  
 private static final Lock *lock* = new ReentrantLock();  
 @Override  
 public int size() {  
 *lock*.lock();  
 int k = map.size();  
 *lock*.unlock();  
 return k;  
 }  
  
 @Override  
 public boolean isEmpty() {  
 *lock*.lock();  
 boolean k = map.isEmpty();  
 *lock*.unlock();  
 return k;  
 }  
  
 @Override  
 public boolean containsKey(Object key) {  
 *lock*.lock();  
 boolean k = map.containsKey(key);  
 *lock*.unlock();  
 return k;  
 }  
  
 @Override  
 public boolean containsValue(Object value) {  
 *lock*.lock();  
 boolean k = map.containsValue(value);  
 *lock*.unlock();  
 return k;  
 }  
  
 @Override  
 public Object get(Object key) {  
 *lock*.lock();  
 Object k = map.get(key);  
 *lock*.unlock();  
 return k;  
 }  
  
 @Override  
 public Object put(Object key, Object value) {  
 *lock*.lock();  
 Object k = map.put(key, value);  
 *lock*.unlock();  
 return k;  
 }  
  
 @Override  
 public Object remove(Object key) {  
 *lock*.lock();  
 Object k = map.remove(key);  
 *lock*.unlock();  
 return k;  
 }  
  
 @Override  
 public void putAll(Map m) {  
 *lock*.lock();  
 map.putAll(m);  
 *lock*.unlock();  
 }  
  
 @Override  
 public void clear() {  
 *lock*.lock();  
 map.clear();  
 *lock*.unlock();  
 }  
  
 @Override  
 public Set keySet() {  
 *lock*.lock();  
 Set k = map.keySet();  
 *lock*.unlock();  
 return k;  
 }  
  
 @Override  
 public Collection values() {  
 *lock*.lock();  
 Collection k = map.values();  
 *lock*.unlock();  
 return k;  
 }  
  
 @Override  
 public Set<Entry> entrySet() {  
 *lock*.lock();  
 Set <Entry> k = map.entrySet();  
 *lock*.unlock();  
 return k;  
 }  
}

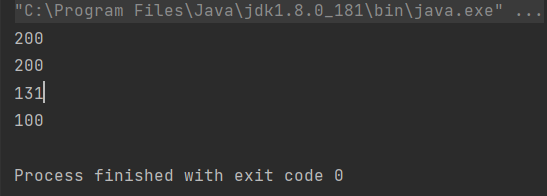
CustomSet.java

import java.util.Collection;  
import java.util.HashSet;  
import java.util.Iterator;  
import java.util.Set;  
  
public class CustomSet implements Set {  
 Set set;  
 public CustomSet(){  
 set = new HashSet();  
 }  
 @Override  
 public synchronized int size() {  
 return set.size();  
 }  
  
 @Override  
 public synchronized boolean isEmpty() {  
 return set.isEmpty();  
 }  
  
 @Override  
 public synchronized boolean contains(Object o) {  
 return set.contains(o);  
 }  
  
 @Override  
 public synchronized Iterator iterator() {  
 return set.iterator();  
 }  
  
 @Override  
 public synchronized Object[] toArray() {  
 return set.toArray();  
 }  
  
 @Override  
 public synchronized boolean add(Object o) {  
 return set.add(o);  
 }  
  
 @Override  
 public synchronized boolean remove(Object o) {  
 return set.remove(o);  
 }  
  
 @Override  
 public synchronized boolean addAll(Collection c) {  
 return set.addAll(c);  
 }  
  
 @Override  
 public synchronized void clear() {  
 set.clear();  
 }  
  
 @Override  
 public synchronized boolean removeAll(Collection c) {  
 return set.removeAll(c);  
 }  
  
 @Override  
 public synchronized boolean retainAll(Collection c) {  
 return set.retainAll(c);  
 }  
  
 @Override  
 public synchronized boolean containsAll(Collection c) {  
 return set.containsAll(c);  
 }  
  
 @Override  
 public synchronized Object[] toArray(Object[] a) {  
 return set.toArray(a);  
 }  
}

Main.java

import java.util.HashMap;  
import java.util.HashSet;  
import java.util.Map;  
import java.util.Set;  
  
public class Main {  
 public static void main(String[] args) throws InterruptedException {  
 Set set = new HashSet();  
 CustomSet set2 = new CustomSet();  
 Map map = new HashMap();  
 CustomMap map2 = new CustomMap();  
 Thread one = new Thread(()->{  
 for(int a = 0; a<100; a++){  
 set.add((Math.*random*() \* ((1000 - 2) + 1)) + 2);  
 set2.add((Math.*random*() \* ((1000 - 2) + 1)) + 2);  
 map.put(a, (Math.*random*() \* ((1000 - 2) + 1)) + 2);  
 map2.put(a, (Math.*random*() \* ((1000 - 2) + 1)) + 2);  
  
 }  
 });  
 Thread two = new Thread(()->{  
 for(int a = 0; a<100; a++){  
 set.add((Math.*random*() \* ((1000 - 2) + 1)) + 2);  
 set2.add((Math.*random*() \* ((1000 - 2) + 1)) + 2);  
 map.put(a, (Math.*random*() \* ((1000 - 2) + 1)) + 2);  
 map2.put(a, (Math.*random*() \* ((1000 - 2) + 1)) + 2);  
 }  
 });  
 one.start();  
 two.start();  
 Thread.*sleep*(3000);  
 System.*out*.println(set.size());  
 System.*out*.println(set2.size());  
 System.*out*.println(map.size());  
 System.*out*.println(map2.size());  
 }  
}

1. Результат работы программы



1. Выводы о проделанной работе

В результате выполнения данной практической работы мы познакомились с принципами построения потокобезопасных коллекций.

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 и пояснения к нему**

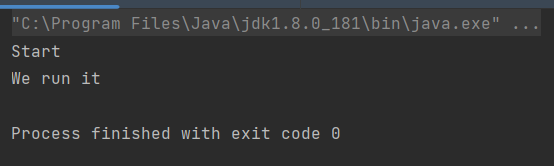
CustomExecutorService.java

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

Main.java

public class Main {  
 public static void main(String[] args) {  
 CustomExecutorService executorService = new CustomExecutorService(2);  
 executorService.submit(() -> {  
 try {  
 Thread.*sleep*(1200);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 System.*out*.println("We run it");  
 });  
 executorService.submit(() -> System.*out*.println("Start"));  
 executorService.shutdown();  
 }  
}

1. **Результат работы программы**



1. **Выводы о проделанной работе**

В результате выполнения данной практической работы мы познакомились с ExecutorService, а также построили собственную его имплементацию.

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 и пояснения к нему**

Singleton.java

public class Singleton {  
 private static Singleton *instance*;  
 private Singleton(){}  
 public static Singleton getInstance(){  
 if(*instance* == null){  
 *instance* = new Singleton();  
 }  
 return *instance*;  
 }  
}

Singleton2.java

public class Singleton2 {  
 private Singleton2(){  
  
 }  
 private static class Singleton2Holder{  
 private static final Singleton2 *instance* = new Singleton2();  
 }  
 public static Singleton2 getInstance(){  
 return Singleton2Holder.*instance*;  
 }  
}

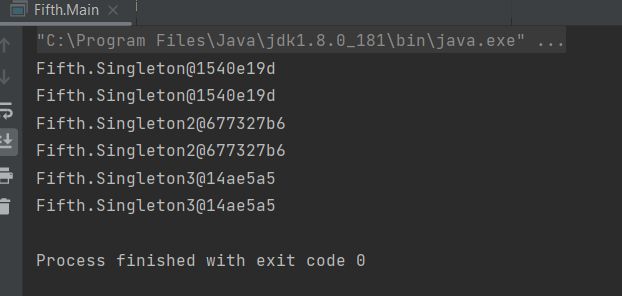
Singleton3,java

public class Singleton3 {  
 private static Singleton3 *instance*;  
 private Singleton3(){}  
 static{  
 try{  
 *instance* = new Singleton3();  
 }  
 catch (Exception e){  
  
 }  
 }  
 public static Singleton3 getInstance(){  
 return *instance*;  
 }  
}

Main.java

public class Main {  
 public static void main(String[] args) {  
 Singleton singleton = Singleton.*getInstance*();  
 Singleton singleton2 = Singleton.*getInstance*();  
 Singleton2 singleton3 = Singleton2.*getInstance*();  
 Singleton2 singleton4 = Singleton2.*getInstance*();  
 Singleton3 singleton5 = Singleton3.*getInstance*();  
 Singleton3 singleton6 = Singleton3.*getInstance*();  
 System.*out*.println(singleton);  
 System.*out*.println(singleton2);  
 System.*out*.println(singleton3);  
 System.*out*.println(singleton4);  
 System.*out*.println(singleton5);  
 System.*out*.println(singleton6);  
 }  
}

1. **Результат работы программы**

****

1. **Выводы о проделанной работе**

В результате выполнения данной практической работы мы познакомились с паттерном синглтон и научились его реализовывать несколькими способами.

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 и пояснения к нему**

AbstractFactory/Chair.java

public interface Chair {  
 void sit();  
}

AbstractFactory/CoffeeTable.java

public interface CoffeeTable {  
 void sit();  
}

AbstractFactory/FurnitureFactory.java

public interface FurnitureFactory {  
 Chair createChair();  
 Sofa createSofa();  
 CoffeeTable createCoffeeTable();  
}

AbstractFactory/Main.java

public class Main {  
 public static void main(String[] args) {  
 ModernFurnitureFactory factory = new ModernFurnitureFactory();  
 Chair chair = factory.createChair();  
 VictorianFurnitureFactory factory1 = new VictorianFurnitureFactory();  
 CoffeeTable table = factory1.createCoffeeTable();  
 Sofa sofa = factory.createSofa();  
 chair.sit();  
 table.sit();  
 sofa.sit();  
 }  
}

AbstractFactory/ModernChair.java

public class ModernChair implements Chair{  
 @Override  
 public void sit() {  
 System.*out*.println("Sitting on Modern chair");  
 }  
}

AbstractFactory/ModernCoffeeTable.java

public class ModernCoffeeTable implements CoffeeTable{  
 @Override  
 public void sit() {  
 System.*out*.println("Sitting on Modern coffee table");  
 }  
}

AbstractFactory/ModernFurnitureFactory.java

public class ModernFurnitureFactory implements FurnitureFactory{  
 @Override  
 public Chair createChair() {  
 return new ModernChair();  
 }  
  
 @Override  
 public Sofa createSofa() {  
 return new ModernSofa();  
 }  
  
 @Override  
 public CoffeeTable createCoffeeTable() {  
 return new ModernCoffeeTable();  
 }  
}

AbstractFactory/ModernSofa.java

public class ModernSofa implements Sofa{  
 @Override  
 public void sit() {  
 System.*out*.println("Sitting on Modern sofa");  
 }  
}

AbstractFactory/Sofa.java

public interface Sofa {  
 void sit();  
}

AbstractFactory/VictorianChair.java

public class VictorianChair implements Chair{  
 @Override  
 public void sit() {  
 System.*out*.println("Sitting on Victorian chair");  
 }  
}

AbstractFactory/VictorianCoffeeTable.java

public class VictorianCoffeeTable implements CoffeeTable{  
 @Override  
 public void sit() {  
 System.*out*.println("Sitting on Victorian coffee table");  
 }  
}

AbstractFactory/VictorianFurnitureFactory.java

public class VictorianFurnitureFactory implements FurnitureFactory{  
  
 @Override  
 public Chair createChair() {  
 return new VictorianChair();  
 }  
  
 @Override  
 public Sofa createSofa() {  
 return new VictorianSofa();  
 }  
  
 @Override  
 public CoffeeTable createCoffeeTable() {  
 return new VictorianCoffeeTable();  
 }  
}

AbstractFactory/VictorianSofa.java

public class VictorianSofa implements Sofa{  
 @Override  
 public void sit() {  
 System.*out*.println("Sitting on Victorian sofa");  
 }  
}

Builder/Builder.java

public interface Builder {  
 void setWallsColor();  
 void setDoorType();  
 void addSwimmingPool();  
 House getResult();  
}

Builder/Director.java

public class Director {  
 Builder builder;  
 public Director(Builder builder){  
 this.builder = builder;  
 createObject();  
 }  
 public void changeBuilder(Builder builder){  
 this.builder = builder;  
 createObject();  
 }  
 private void createObject(){  
 builder.addSwimmingPool();  
 builder.setDoorType();  
 builder.setWallsColor();  
 }  
 public void printResult(){  
 System.*out*.println(builder.getResult().toString());  
 }  
}

Builder/FirstHouseBuilder.java

public class FirstHouseBuilder implements Builder{  
 House house;  
 public FirstHouseBuilder(){  
 house = new House();  
 }  
 @Override  
 public void setWallsColor() {  
 house.setWallsColor("blue");  
 }  
  
 @Override  
 public void setDoorType() {  
 house.setDoorType("oak");  
 }  
  
 @Override  
 public void addSwimmingPool() {  
 house.setPool("");  
 }  
  
 public House getResult(){  
 return house;  
 }  
}

Builder/House.java

public class House {  
 String wallsColor;  
 String doorType;  
 String pool;  
  
 public String getWallsColor() {  
 return wallsColor;  
 }  
  
 public void setWallsColor(String wallsColor) {  
 this.wallsColor = wallsColor;  
 }  
  
 public String getDoorType() {  
 return doorType;  
 }  
  
 public void setDoorType(String doorType) {  
 this.doorType = doorType;  
 }  
  
 public String getPool() {  
 return pool;  
 }  
  
 public void setPool(String pool) {  
 this.pool = pool;  
 }  
  
 @Override  
 public String toString() {  
 return "House{" +  
 "wallsColor='" + wallsColor + '\'' +  
 ", doorType='" + doorType + "\' " + pool +  
 '}';  
 }  
}

Builder/Main.java

public class Main {  
 public static void main(String[] args) {  
 Builder builder1 = new FirstHouseBuilder();  
 Builder builder2 = new SecondHouseBuilder();  
 Director director1 = new Director(builder1);  
 Director director2 = new Director(builder2);  
 director1.printResult();  
 director2.printResult();  
 }  
  
}

Builder/SecondHouseBuilder.java

public class SecondHouseBuilder implements Builder{  
 House house;  
 public SecondHouseBuilder(){  
 house = new House();  
 }  
 @Override  
 public void setWallsColor() {  
 house.setWallsColor("red");  
 }  
  
 @Override  
 public void setDoorType() {  
 house.setDoorType("birch");  
 }  
  
 @Override  
 public void addSwimmingPool() {  
 house.setPool("pool included");  
 }  
  
 public House getResult(){  
 return house;  
 }  
}

FabricMethod/Logistics.java

public class Logistics {  
 Transport transport;  
  
 void createTransport(int code){  
 if(code == 1){  
 transport = new Truck();  
 }  
 else if(code == 2){  
 transport = new Ship();  
 }  
 }  
  
 public Transport getTransport() {  
 return transport;  
 }  
  
}

FabricMethod/Main.java

public class Main {  
 public static void main(String[] args) {  
 Logistics logistics = new Logistics();  
 logistics.createTransport(1);  
 Logistics logistics2 = new Logistics();  
 logistics2.createTransport(2);  
 logistics.getTransport().deliver();  
 logistics2.getTransport().deliver();  
 }  
}

FabricMethod/Ship.java

public class Ship implements Transport{  
 @Override  
 public void deliver() {  
 System.*out*.println("Delivered by sea");  
 }  
}

FabricMethod/Transport.java

public interface Transport {  
 void deliver();  
}

FabricMethod/Truck.java

public class Truck implements Transport{  
 public void deliver() {  
 System.*out*.println("Delivered by land");  
 }  
}

Prototype/Circle.java

public class Circle implements Shape{  
 private int radius;  
 private String color;  
  
 public Circle(int radius, String color) {  
 this.radius = radius;  
 this.color = color;  
 }  
  
 public Circle(Circle circle){  
 this.radius = circle.getRadius();  
 this.color = circle.getColor();  
 }  
  
 @Override  
 public Shape clone() {  
 return new Circle(this);  
 }  
  
 public int getRadius() {  
 return radius;  
 }  
  
 public void setRadius(int radius) {  
 this.radius = radius;  
 }  
  
 public String getColor() {  
 return color;  
 }  
  
 public void setColor(String color) {  
 this.color = color;  
 }  
  
 @Override  
 public String toString() {  
 return "Circle{" +  
 "radius=" + radius +  
 ", color='" + color + '\'' +  
 '}';  
 }  
}

Prototype/Main.java

public class Main {  
 public static void main(String[] args) {  
 Shape rectangle = new Rectangle(5, 4, "red");  
 Shape circle = new Circle(3, "blue");  
 Shape newCircle = circle.clone();  
 Shape newRectangle = rectangle.clone();  
 System.*out*.println(rectangle.toString());  
 System.*out*.println(newRectangle.toString());  
 System.*out*.println(circle.toString());  
 System.*out*.println(newCircle.toString());  
 }  
}

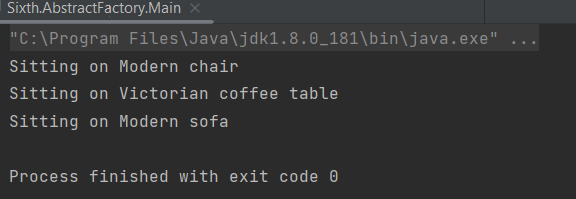
Prototype/Rectangle.java

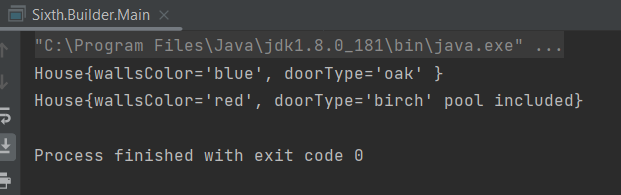
public class Rectangle implements Shape{  
 private int width;  
 private int height;  
 private String color;  
  
 public Rectangle(int width, int height, String color) {  
 this.width = width;  
 this.height = height;  
 this.color = color;  
 }  
  
 public Rectangle(Rectangle rectangle){  
 this.color = rectangle.getColor();  
 this.height = rectangle.getHeight();  
 this.width = rectangle.getWidth();  
 }  
  
 public int getWidth() {  
 return width;  
 }  
  
 public void setWidth(int width) {  
 this.width = width;  
 }  
  
 public int getHeight() {  
 return height;  
 }  
  
 public void setHeight(int height) {  
 this.height = height;  
 }  
  
 public String getColor() {  
 return color;  
 }  
  
 public void setColor(String color) {  
 this.color = color;  
 }  
  
 @Override  
 public String toString() {  
 return "Rectangle{" +  
 "width=" + width +  
 ", height=" + height +  
 ", color='" + color + '\'' +  
 '}';  
 }  
  
 @Override  
 public Shape clone() {  
 return new Rectangle(this);  
 }  
}

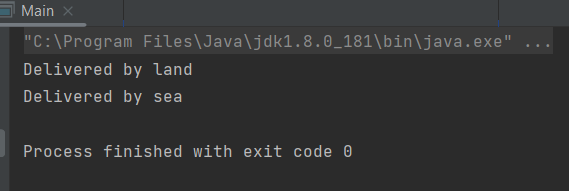
Prototype/Shape.java

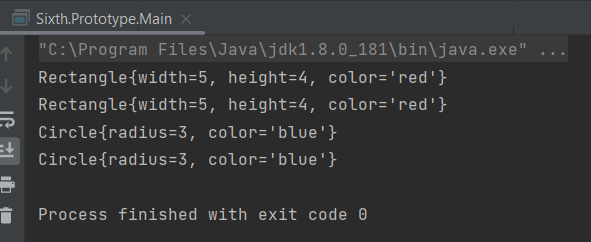
public interface Shape {  
 Shape clone();  
}

1. **Результат работы программы:**

****

****

****

****

1. **Выводы о проделанной работе**

В результате выполнения данной практической работы мы познакомились с порождающими паттернами проектирования и реализовали их.

Практическая работа №7

1. **Постановка задачи и персональный вариант**

Тема: Реализация структурных паттернов проектирования

Постановка задачи: Написать реализацию паттерна в соответствии с вариантом

индивидуального задания. Компоновщик, Декоратор.

1. **Код и пояснения к нему**

Composite/Component.java

import java.util.ArrayList;  
  
public class Component implements Composite{  
 private ArrayList<Composite> componentList;  
  
 public Component() {  
 componentList = new ArrayList<>();  
 }  
  
 public void addToList(Composite object){  
 componentList.add(object);  
 }  
  
 @Override  
 public double execute() {  
 double result = 0;  
 for (Composite composite:componentList) {  
 result+=composite.execute();  
 }  
 return result;  
 }  
}

Composite/Composite.java

public interface Composite {  
 double execute();  
}

Composite/Leaf.java

public class Leaf implements Composite{  
 private double price;  
  
 public Leaf(double price) {  
 this.price = price;  
 }  
  
 @Override  
 public double execute() {  
 return price;  
 }  
  
 public void setPrice(double price) {  
 this.price = price;  
 }  
}

Composite/Main.java

public class Main {  
 public static void main(String[] args) {  
 Component main = new Component();  
 Component box1 = new Component();  
 Component box2 = new Component();  
 Component box3 = new Component();  
 Component box4 = new Component();  
 Component box5 = new Component();  
 Leaf leaf1 = new Leaf(1.2);  
 Leaf leaf2 = new Leaf(2.3);  
 Leaf leaf3 = new Leaf(3.4);  
 Leaf leaf4 = new Leaf(4.5);  
 Leaf leaf5 = new Leaf(5.6);  
 Leaf leaf6 = new Leaf(6.7);  
 Leaf leaf7 = new Leaf(7.8);  
 Leaf leaf8 = new Leaf(8.9);  
 Leaf leaf9 = new Leaf(9.1);  
 Leaf leaf10 = new Leaf(10.2);  
 Leaf leaf11 = new Leaf(11.3);  
 Leaf leaf12 = new Leaf(12.4);  
 Leaf leaf13 = new Leaf(13.5);  
 Leaf leaf14 = new Leaf(14.6);  
 Leaf leaf15 = new Leaf(15.7);  
 box1.addToList(leaf1);  
 box1.addToList(leaf2);  
 box1.addToList(leaf3);  
 box2.addToList(leaf4);  
 box2.addToList(leaf5);  
 box2.addToList(leaf6);  
 box3.addToList(leaf7);  
 box3.addToList(leaf8);  
 box3.addToList(leaf9);  
 box4.addToList(leaf10);  
 box4.addToList(leaf11);  
 box4.addToList(leaf12);  
 box5.addToList(leaf13);  
 box5.addToList(leaf14);  
 box5.addToList(leaf15);  
 box1.addToList(box2);  
 box1.addToList(box3);  
 box2.addToList(box4);  
 box3.addToList(box5);  
 main.addToList(box1);  
 System.*out*.println(main.execute());  
 }  
}

Decorator/Car.java

public interface Car {  
 int getSpeed();  
 int getBaggageWeight();  
}

Decorator/Main.java

public class Main {  
 public static void main(String[] args) {  
 SimpleCar car = new SimpleCar();  
 System.*out*.println(car.getBaggageWeight() + " " + car.getSpeed());  
 Car sportCar = new SportCar(car);  
 System.*out*.println(sportCar.getBaggageWeight() + " " + sportCar.getSpeed());  
 Car truck = new Truck(car);  
 System.out.println(truck.getBaggageWeight() + " " + truck.getSpeed());  
 }  
}

Decorator/SimpleCar.java

public class SimpleCar implements Car{  
 private int speed = 50;  
 private int baggageWeight = 100;  
  
 @Override  
 public int getSpeed() {  
 return speed;  
 }  
  
 @Override  
 public int getBaggageWeight() {  
 return baggageWeight;  
 }  
  
}

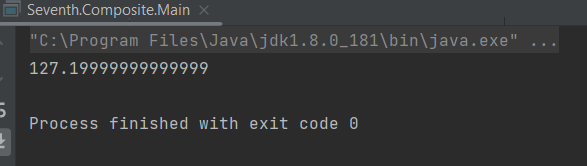
Decorator/SportCar.java

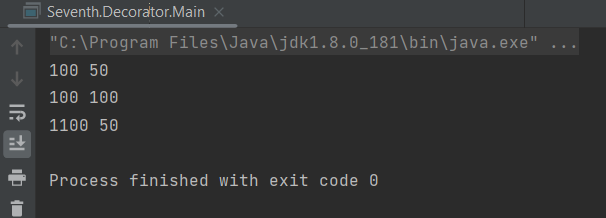
public class SportCar implements Car{  
 private Car car;  
 public SportCar(Car car){  
 this.car = car;  
 }  
  
 @Override  
 public int getSpeed() {  
 return this.car.getSpeed() + 50;  
 }  
  
 @Override  
 public int getBaggageWeight() {  
 return this.car.getBaggageWeight();  
 }  
}

Decorator/Truck.java

public class Truck implements Car{  
 private Car car;  
 public Truck(Car car){  
 this.car = car;  
 }  
  
 @Override  
 public int getSpeed() {  
 return this.car.getSpeed();  
 }  
  
 @Override  
 public int getBaggageWeight() {  
 return this.car.getBaggageWeight() + 1000;  
 }  
  
}

1. **Результат работы программы**

****

****

1. **Выводы о проделанной работе**

В результате выполнения данной практической работы мы познакомились с структурными паттернами и реализовали их.

Практическая работа №8

1. **Постановка задачи и персональный вариант**

Тема: Реализация поведенческих паттернов проектирования.Постановка задачи: Написать реализацию паттерна в соответствии с вариантом

индивидуального задания. Итератор, Посредник

1. **Код и пояснения к нему**

Iterator/Collection.java

import java.util.ArrayList;  
  
public class Collection implements IterableCollection{  
 private ArrayList<Integer> list = new ArrayList<>();  
  
 public ArrayList<Integer> getList() {  
 return list;  
 }  
  
 public void setList(ArrayList<Integer> list) {  
 this.list = list;  
 }  
  
 public void addToList(int number){  
 list.add(number);  
 }  
  
 @Override  
 public Iterator createIterator(int code) {  
 switch (code){  
 case 1:  
 return new Iterator1(list);  
 case 2:  
 return new Iterator2(list);  
 default:  
 return null;  
 }  
 }  
}

Iterator/IterableConnection,java

public interface IterableCollection {  
 Iterator createIterator(int code);  
}

Iterator/Iterator.java

public interface Iterator {  
 boolean hasNext();  
 int getNext();  
}

Iterator/Iterator1.java

import java.util.ArrayList;  
  
public class Iterator1 implements Iterator{  
 private ArrayList<Integer> list;  
 private int currentPos = 0;  
  
 public Iterator1(ArrayList<Integer> list) {  
 this.list = list;  
 }  
  
 @Override  
 public boolean hasNext() {  
 if(list.size()>currentPos){  
 return true;  
 }  
 return false;  
 }  
  
 @Override  
 public int getNext() {  
 int result = list.get(currentPos);  
 currentPos++;  
 return result;  
 }  
}

Iterator/Iterator2.java

import java.util.ArrayList;  
  
public class Iterator2 implements Iterator{  
 private ArrayList<Integer> list;  
 private int currentPos = 0;  
  
 public Iterator2(ArrayList<Integer> list) {  
 this.list = list;  
 }  
  
 @Override  
 public boolean hasNext() {  
 if(list.size()>currentPos+1){  
 return true;  
 }  
 return false;  
 }  
  
 @Override  
 public int getNext() {  
 int result = list.get(currentPos);  
 currentPos+=2;  
 return result;  
 }  
}

Iterator/Main.java

public class Main {  
 public static void main(String[] args) {  
 Collection collection = new Collection();  
 Collection collection1 = new Collection();  
 Iterator it1 = collection.createIterator(1);  
 Iterator it2 = collection1.createIterator(2);  
 for(int a = 0; a<20; a++){  
 collection.addToList(a);  
 collection1.addToList(a);  
 }  
 while(it1.hasNext()){  
 System.*out*.println(it1.getNext());  
 }  
 while (it2.hasNext()){  
 System.*out*.println(it2.getNext());  
 }  
 }  
}

Mediator/Button.java

public class Button implements Components{  
 public Mediator mediator;  
  
 private String name;  
  
 public Button(String name) {  
 this.name = name;  
 mediator = new ConcreteMediator();  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public void onPress() {  
 mediator.notify(this);  
 }  
}

Mediator/Checkbox.java

public class Checkbox implements Components{  
 public Mediator mediator;  
  
 String name;  
  
 public Checkbox(String name) {  
 this.name = name;  
 mediator = new ConcreteMediator();  
 }  
  
 public String getName() {  
 return name;  
 }  
  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public void onPress() {  
 mediator.notify(this);  
 }  
}

Mediator/Components.java

public interface Components {  
 String getName();  
 void onPress();  
}

Mediator/ConcreteMediator.java

public class ConcreteMediator implements Mediator{  
 @Override  
 public void notify(Components components) {  
 System.*out*.println("User pressed " + components.getName());  
 }  
}

Mediator/Main.java

public class Main {  
 public static void main(String[] args) {  
 Components button1 = new Button("Button1");  
 Components button2 = new Button("Button2");  
 Components checkbox1 = new Checkbox("Checkbox1");  
 Components radioButton1 = new RadioButton("Radiobutton1");  
 Components checkbox2 = new Checkbox("Checkbox2");  
 button1.onPress();  
 button2.onPress();  
 checkbox1.onPress();  
 checkbox2.onPress();  
 radioButton1.onPress();  
  
 }  
}

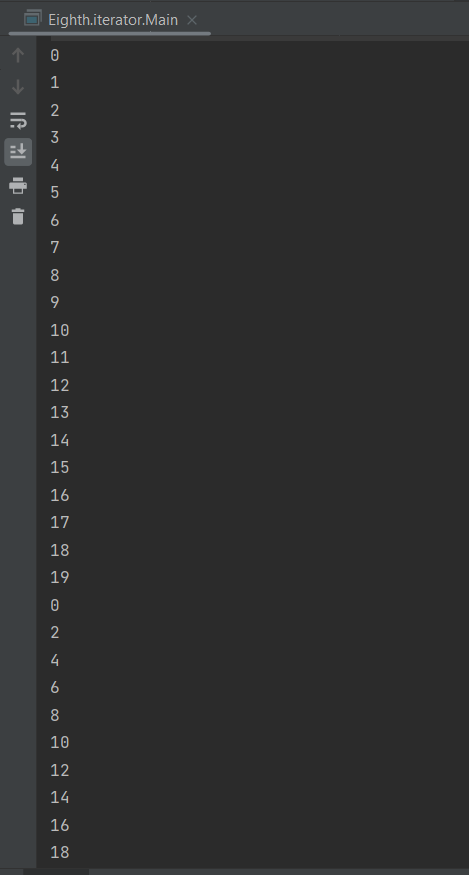
Mediator/Mediator.java

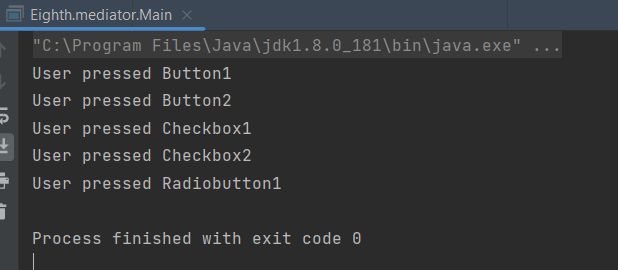
public interface Mediator {  
 void notify(Components components);  
}

Mediator/RadioButton.java

public class RadioButton implements Components {  
 public Mediator mediator;  
  
 String name;  
  
 public RadioButton(String name) {  
 this.name = name;  
 mediator = new ConcreteMediator();  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public void onPress() {  
 mediator.notify(this);  
 }  
}

1. **Результат работы программы**

****

****

1. **Выводы о проделанной работе**

В результате выполнения данной практической работы мы познакомились с поведенчискеми паттернами и реализовали их.

**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 и пояснения к нему**

Build.gradle

plugins **{** id 'java'  
**}**group 'org.example'  
version '1.0-SNAPSHOT'  
  
repositories **{** mavenCentral()  
**}**dependencies **{** testImplementation 'org.junit.jupiter:junit-jupiter-api:5.6.0'  
 testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine'  
**}**test **{** useJUnitPlatform()  
**}**jar **{** manifest **{** attributes(  
 'Main-Class' : 'ru.mirea.Main'  
 )  
 **}  
}**task copyJar(type: Copy) **{** dependsOn 'jar'  
 from file("build/libs/ninth1-1.0-SNAPSHOT.jar")  
 into file("docker/")  
**}**task buildImage(type: Exec) **{** dependsOn 'copyJar'  
 commandLine 'docker', 'build', '-t', 'task9', './docker'  
**}**task runDocker(type: Exec) **{** dependsOn 'buildImage'  
 commandLine 'docker', 'run', '--name', 'task9', 'task9'  
**}**

Dockerfile

FROM gradle  
ADD . .  
CMD ["java","-jar","ninth1-1.0-SNAPSHOT.jar"]

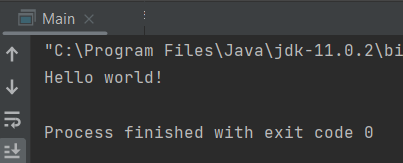
Docker-compose.yml

version: "3.8"  
services:  
 server:  
 build:  
 context: .  
 ports:  
 - 8080:80

Main.java

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

1. **Результат работы программы**



1. **Выводы о проделанной работе**

В результате выполнения данной практической работы мы познакомились с системой сборки Gradle и научились на её основе создавать проекты.

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 определяются в соответствии с вариантом индивидуального задания.

1. **Код и пояснения к нему**

BeanConfig.java

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 Junior junior(){  
 return new Junior();  
 }  
 @Bean  
 public Middle middle(){  
 return new Middle();  
 }  
 @Bean  
 public Senior senior(){  
 return new Senior();  
 }  
}

Junior.java

import org.springframework.stereotype.Component;  
  
@Component  
public class Junior implements Programmer{  
 @Override  
 public void doCoding() {  
 System.*out*.println("java.lang.NullPointerException");  
 }  
}

Main.java

import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public static void main(String[] args) {  
 ApplicationContext context = new AnnotationConfigApplicationContext(BeanConfig.class);  
 switch (args[0]) {  
 case "Junior":  
 Junior junior = context.getBean(Junior.class);  
 junior.doCoding();  
 break;  
 case "Middle":  
 Middle middle = context.getBean(Middle.class);  
 middle.doCoding();  
 break;  
 case "Senior":  
 Senior senior = context.getBean(Senior.class);  
 senior.doCoding();  
 break;  
 }  
 }  
}

Middle.java

import org.springframework.stereotype.Component;  
  
@Component  
public class Middle implements Programmer{  
 @Override  
 public void doCoding() {  
 System.*out*.println("Well... it works");  
 }  
}

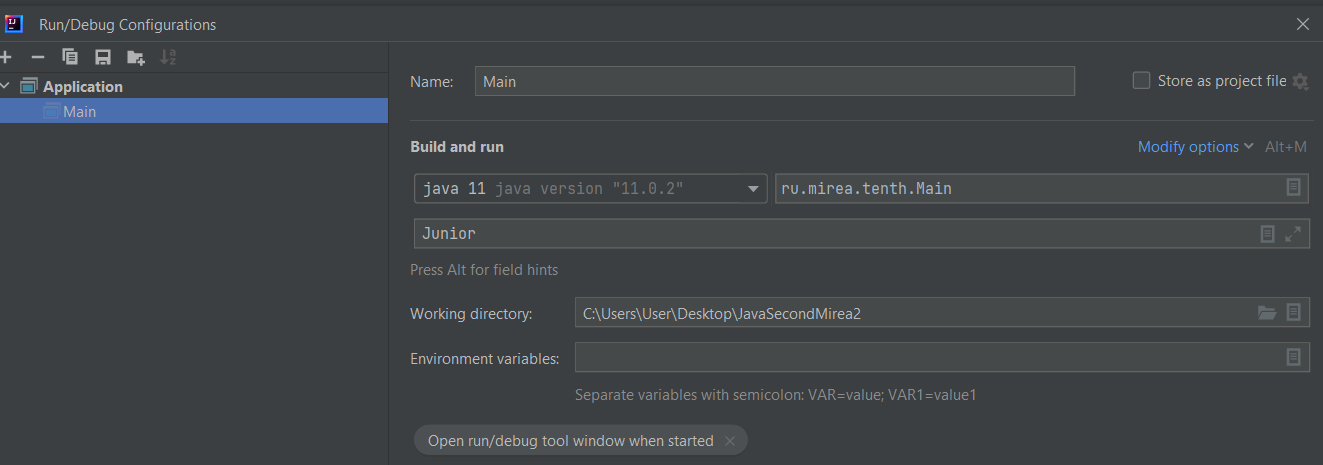
Programmer.java

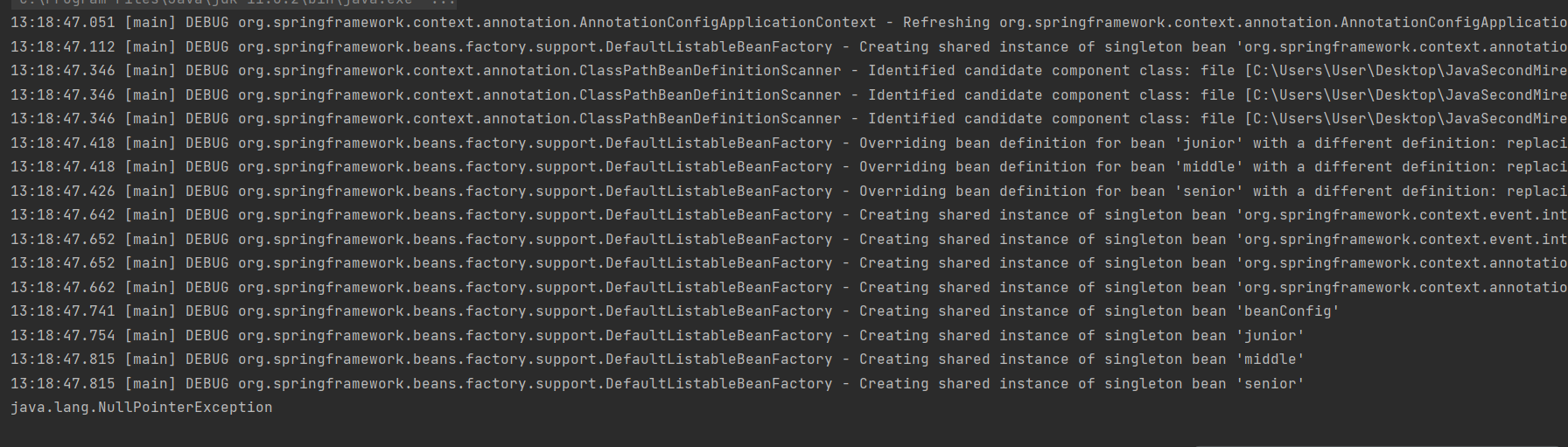
public interface Programmer {  
 void doCoding();  
}

Senior.java

import org.springframework.stereotype.Component;  
  
@Component  
public class Senior implements Programmer{  
  
 @Override  
 public void doCoding() {  
 System.*out*.println("Perfecto!");  
 }  
}

1. **Результат работы программы**

****

****

1. **Выводы о проделанной работе**

В результате выполнения данной практической работы мы познакомились с Spring framework-ом и принципом инверсии контроля. Создали на его основе приложение

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 и пояснения к нему**

Chair.java

import lombok.Getter;  
import lombok.Setter;  
  
@Getter  
@Setter  
public class Chair {  
 private long id;  
 private double price;  
 private String type;  
  
 public Chair(long id, double price, String type) {  
 this.id = id;  
 this.price = price;  
 this.type = type;  
 }  
}

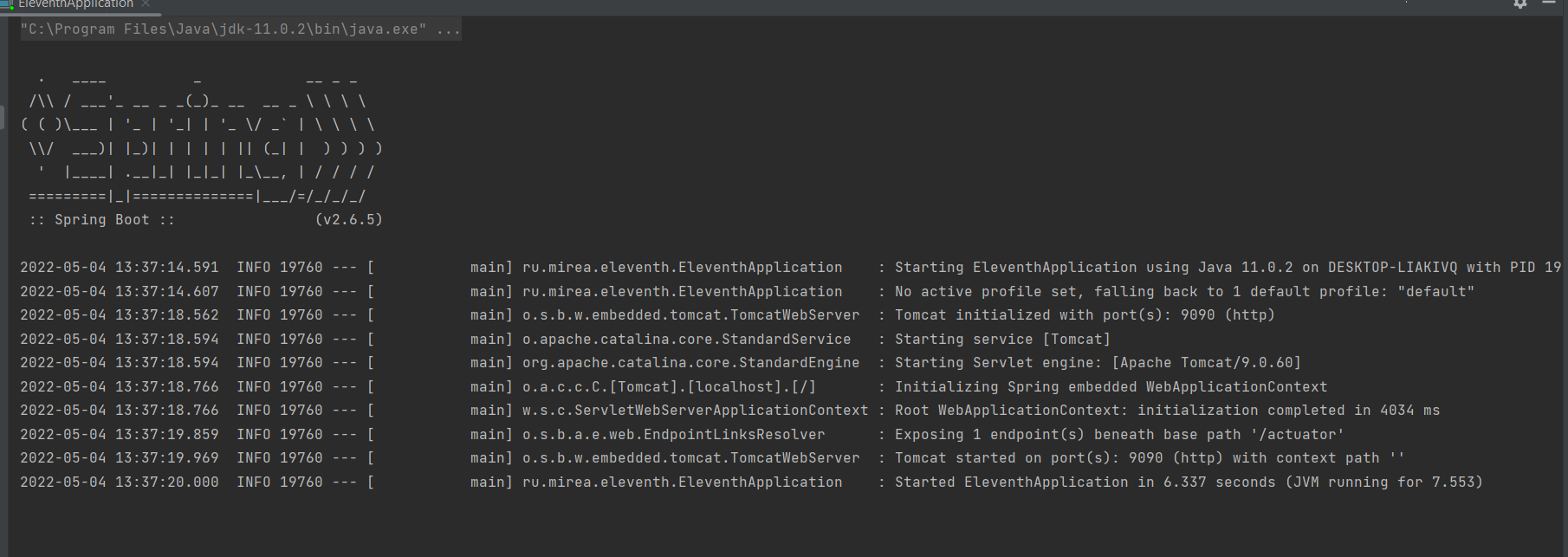
ChairController.java

import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.concurrent.atomic.AtomicLong;  
  
@RestController  
public class ChairController {  
 private final AtomicLong counter = new AtomicLong();  
  
 @GetMapping("/chair")  
 public Chair chair(@RequestParam(value = "price", defaultValue = "0") String price, @RequestParam(value = "type", defaultValue = "0") String name ){  
 return new Chair(counter.incrementAndGet(), Long.*parseLong*(price), name);  
 }  
 //http://localhost:8080/chair  
}

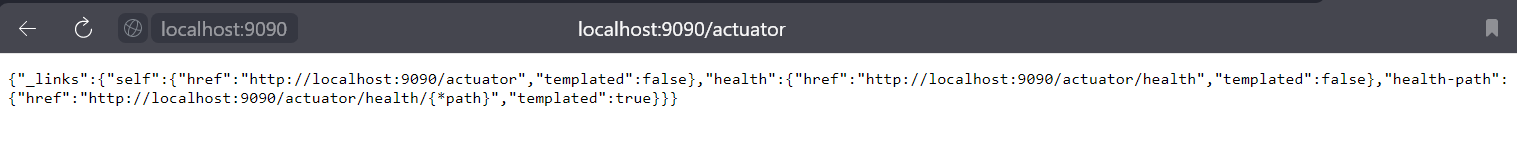
EleventhApplication.java

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

1. **Результат работы программы**

****

****

****

1. **Выводы о проделанной работе**

В результате выполнения данной практической работы мы познакомились с spring boot initializer, созданием веб приложений и Lombok.

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 и пояснения к нему**

BeanConfig.java

import org.springframework.context.annotation.ComponentScan;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
@ComponentScan  
public class BeanConfig {  
}

Controller.java

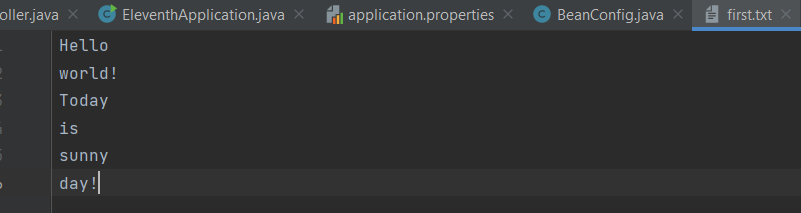
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.stereotype.Component;  
  
import javax.annotation.PostConstruct;  
import java.io.File;  
import java.io.FileReader;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.util.HashSet;  
  
@Component  
public class Controller {  
 private static Logger *LOG* = LoggerFactory  
 .*getLogger*(Main.class);  
 public void doMagic(String[] args) {  
 String one = "C:\\Users\\User\\Desktop\\eleventh\\src\\main\\java\\ru\\mirea\\twelfth\\" + args[0];  
 String two = "C:\\Users\\User\\Desktop\\eleventh\\src\\main\\java\\ru\\mirea\\twelfth\\" + args[1];  
 FileWriter writer = null;  
 File file = new File(two);  
 try {  
 writer = new FileWriter(file, false);  
 }  
 catch (Exception ex){  
 *LOG*.info(ex.getMessage());  
 }  
 try (FileReader reader = new FileReader(one)){  
 Path path = Path.*of*(one);  
 HashSet set = new HashSet(Files.*readAllLines*(path));  
 for (Object k: set  
 ) {  
 *LOG*.info(k.toString());  
 writer.write(k.toString());  
 writer.write("\n");  
 }  
 writer.flush();  
 }  
 catch (Exception ex){  
 try {  
 writer.write("NULL");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 try {  
 writer.flush();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 @PostConstruct  
 public void init(){  
 *LOG*.info("Controller is ready!");  
 }  
}

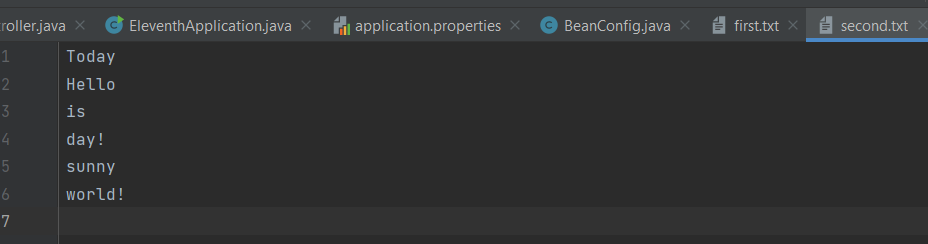
Main.java

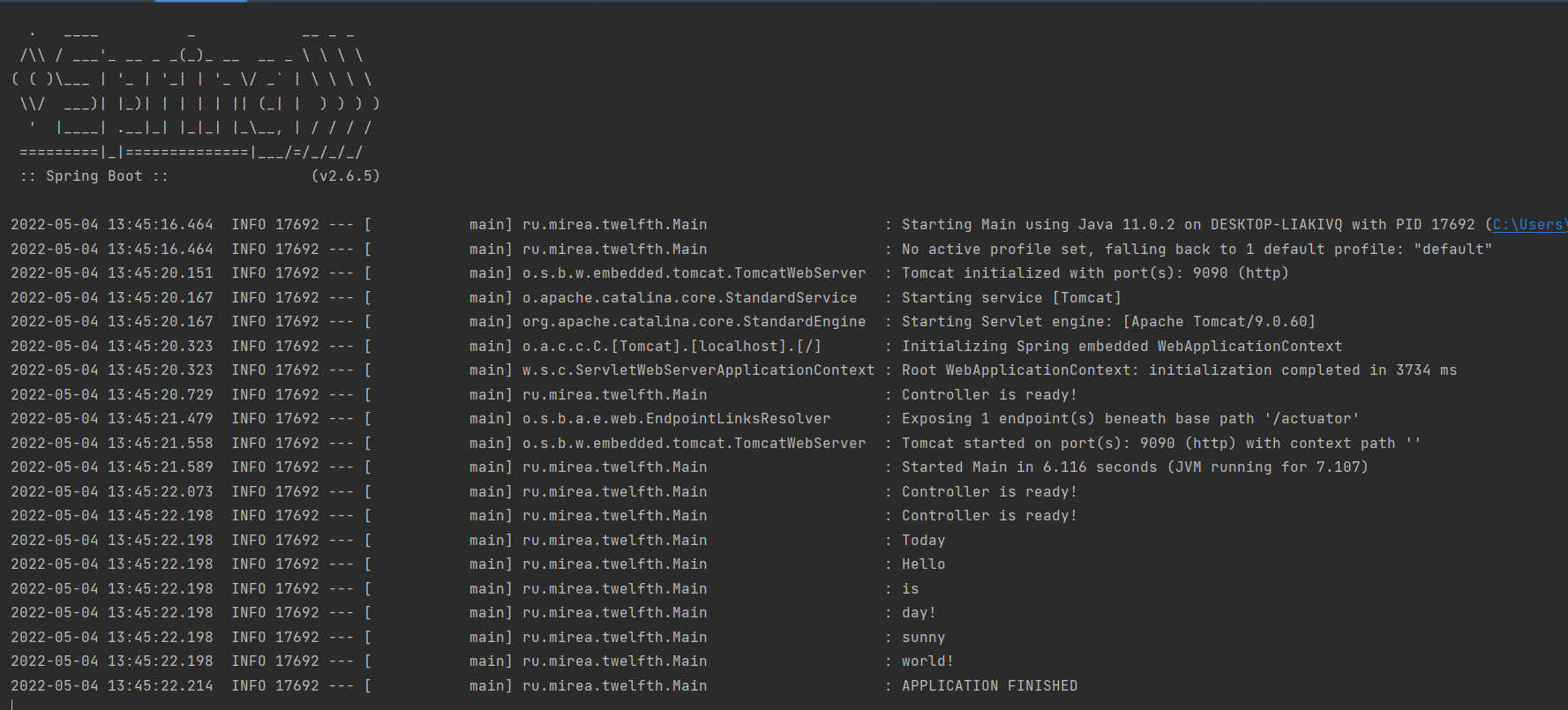
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.boot.CommandLineRunner;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

@SpringBootApplication  
public class Main implements CommandLineRunner {  
 private static Logger *LOG* = LoggerFactory  
 .*getLogger*(Main.class);  
  
 public static void main(String[] args) {  
 *LOG*.info("STARTING THE APPLICATION");  
 SpringApplication.*run*(Main.class, args);  
 *LOG*.info("APPLICATION FINISHED");  
 }  
  
  
 @Override  
 public void run(String... args) {  
 ApplicationContext context = new AnnotationConfigApplicationContext(BeanConfig.class);  
 Controller controller = context.getBean(Controller.class);  
 controller.init();  
 controller.doMagic(args);  
 }  
}

1. **Результат работы программы**







1. **Выводы о проделанной работе**

В результаты выполнения данной практической работы мы познакомились с аннотациями @PostConstruct и @PreDestroy и построили на их основе приложение.

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 программы и пояснения к нему**

BeanConfig.java

import org.springframework.context.annotation.ComponentScan;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
@ComponentScan  
public class BeanConfig {  
}

Main.java

import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
  
import javax.annotation.PreDestroy;  
  
@SpringBootApplication  
public class Main {  
  
  
 public static void main(String[] args) {  
 ApplicationContext context = new AnnotationConfigApplicationContext(BeanConfig.class);  
 Student student = context.getBean(Student.class);  
 System.out.println(student.toString());  
 }  
}

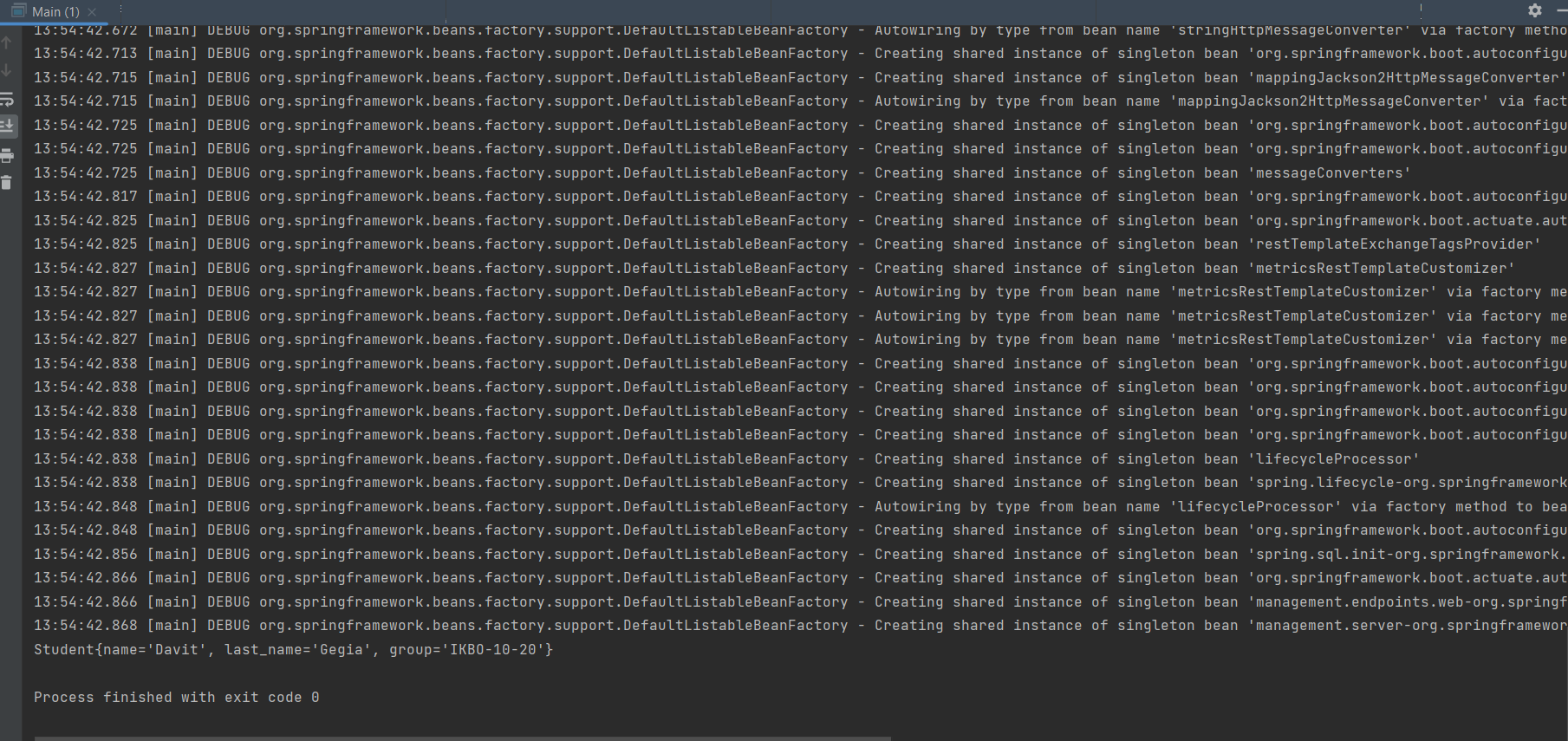
Student.java

import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
import javax.annotation.PostConstruct;  
  
@Component  
public class Student {  
 @Value("${program.student.name}")  
 private String name;  
 @Value("${program.student.last\_name}")  
 private String last\_name;  
 @Value("${program.student.group}")  
 private String group;  
  
 @PostConstruct  
 @Override  
 public String toString() {  
 return "Student{" +  
 "name='" + name + '\'' +  
 ", last\_name='" + last\_name + '\'' +  
 ", group='" + group + '\'' +  
 '}';  
 }  
  
  
}

Application.yml

program:  
 student:  
 name: Davit  
 last\_name: Gegia  
 group: IKBO-10-20

1. **Результат работы программы**

****

1. **Выводы о проделанной работе**

В результате выполнения данной практической работы мы познакомились с интерфейсом Environment и аннотацией Value.

Практическая работа №14

1. **Постановка задачи и персональный вариант**

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

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

Создать класс Book с полями name, creationDate. Создать класс Author с полями firstName, lastName, middleName, birthDate. Создать классыконтроллеры для создания, удаления объектов и получения всех объектов каждого типа. Сами объекты хранить в памяти.

1. **Код и пояснения к нему**

Author.java

import lombok.Getter;  
import lombok.Setter;  
import lombok.ToString;  
  
@Getter  
@Setter  
@ToString  
public class Author {  
 private String firstName;  
 private String lastName;  
 private String middleName;  
 private String birthDate;  
  
 public Author(String firstName, String lastName, String middleName, String birthDate) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.middleName = middleName;  
 this.birthDate = birthDate;  
 }  
  
}

AuthorController.java

import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.web.bind.annotation.\*;  
import ru.mirea.twelfth.Main;  
  
import java.io.File;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.util.ArrayList;  
import java.util.HashSet;  
  
@RestController  
public class AuthorController {  
 Path path = Path.*of*("C:\\Users\\User\\Desktop\\eleventh\\src\\main\\java\\ru\\mirea\\fourteenth\\authors.txt");  
 File file = new File("C:\\Users\\User\\Desktop\\eleventh\\src\\main\\java\\ru\\mirea\\fourteenth\\authors.txt");  
 private static Logger *LOG* = LoggerFactory  
 .*getLogger*(Main.class);  
 @GetMapping("/author")  
 public String authorGet(@RequestParam(value = "firstName", defaultValue = "null") String name, @RequestParam(value = "all", defaultValue = "false") String all){  
 String list = "";  
 try {  
 HashSet set = new HashSet(Files.*readAllLines*(path));  
 for (Object k: set  
 ) {  
 list+=(k.toString());  
 list += "\n";  
 if(all == "false") {  
 if (k.toString().contains(name)) {  
 return k.toString();  
 }  
 }  
 }  
 if(all.equals("true")){  
 return list;  
 }  
 return "null";  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
  
 @PostMapping("/author")  
 public String authorPost(@RequestParam(value = "firstName", defaultValue = "null") String firstName, @RequestParam(value = "lastName", defaultValue = "null") String lastName, @RequestParam(value = "middleName", defaultValue = "null") String middleName, @RequestParam(value = "birthDate", defaultValue = "null") String birthDate){  
 try {  
 FileWriter writer = new FileWriter(file, true);  
 Author author = new Author(firstName, lastName, middleName, birthDate);  
 writer.write(author.toString() + "\n");  
 writer.flush();  
 return "Success!";  
 } catch (IOException e) {  
 e.printStackTrace();  
 return "Error!";  
 }  
 }  
  
 @DeleteMapping("/author")  
 public String bookDel(@RequestParam(value = "name", defaultValue = "null") String name){  
 if(name == "null"){  
 return "null";  
 }  
  
 try {  
 HashSet set = new HashSet(Files.*readAllLines*(path));  
 ArrayList<String> list = new ArrayList<>();  
 for (Object k: set  
 ) {  
 if(k.toString().contains(name)){  
 continue;  
 }  
 list.add(k.toString());  
 }  
 try {  
 FileWriter writer = new FileWriter(file, false);  
 for (String k: list  
 ) {  
 writer.write(k + "\n");  
 }  
 writer.flush();  
 return "Success!";  
 } catch (IOException e) {  
 e.printStackTrace();  
 return "Error!";  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
}

Book.java

import lombok.Getter;  
import lombok.Setter;  
import lombok.ToString;  
  
@Getter  
@Setter  
@ToString  
public class Book {  
 private String name;  
 private String creationDate;  
  
 public Book(String name, String creationDate) {  
 this.name = name;  
 this.creationDate = creationDate;  
 }  
}

BookController.java

import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.\*;  
import ru.mirea.twelfth.Main;  
  
import java.io.\*;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.util.ArrayList;  
import java.util.HashSet;  
  
@RestController  
public class BookController {  
 Path path = Path.*of*("C:\\Users\\User\\Desktop\\eleventh\\src\\main\\java\\ru\\mirea\\fourteenth\\books.txt");  
 File file = new File("C:\\Users\\User\\Desktop\\eleventh\\src\\main\\java\\ru\\mirea\\fourteenth\\books.txt");  
 private static Logger *LOG* = LoggerFactory  
 .*getLogger*(Main.class);  
 @GetMapping("/book")  
 public String bookGet(@RequestParam(value = "name", defaultValue = "null") String name, @RequestParam(value = "all", defaultValue = "false") String all){  
 String list = "";  
 try {  
 HashSet set = new HashSet(Files.*readAllLines*(path));  
 for (Object k: set  
 ) {  
 list+=(k.toString());  
 list += "\n";  
 if(all == "false") {  
 if (k.toString().contains(name)) {  
 return k.toString();  
 }  
 }  
 }  
 if(all.equals("true")){  
 return list;  
 }  
 return "null";  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
  
 @PostMapping("/book")  
 public String bookPost(@RequestParam(value = "name", defaultValue = "null") String name, @RequestParam(value = "creationDate", defaultValue = "null") String creationDate){  
 try {  
 FileWriter writer = new FileWriter(file, true);  
 Book book = new Book(name, creationDate);  
 writer.write(book.toString() + "\n");  
 writer.flush();  
 return "Success!";  
 } catch (IOException e) {  
 e.printStackTrace();  
 return "Error!";  
 }  
 }  
  
 @DeleteMapping("/book")  
 public String bookDel(@RequestParam(value = "name", defaultValue = "null") String name){  
 if(name == "null"){  
 return "null";  
 }  
  
 try {  
 HashSet set = new HashSet(Files.*readAllLines*(path));  
 ArrayList <String> list = new ArrayList<>();  
 for (Object k: set  
 ) {  
 if(k.toString().contains(name)){  
 continue;  
 }  
 list.add(k.toString());  
 }  
 try {  
 FileWriter writer = new FileWriter(file, false);  
 for (String k: list  
 ) {  
 writer.write(k + "\n");  
 }  
 writer.flush();  
 return "Success!";  
 } catch (IOException e) {  
 e.printStackTrace();  
 return "Error!";  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
}

HomeController.java

import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.ResponseBody;  
import org.springframework.web.bind.annotation.RestController;  
import org.springframework.web.servlet.ModelAndView;  
  
@Controller  
public class HomeController {  
 @GetMapping("/home")  
 @ResponseBody  
 public ModelAndView html(){  
 ModelAndView modelAndView = new ModelAndView();  
 modelAndView.setViewName("student.html");  
 return modelAndView;  
 }  
}

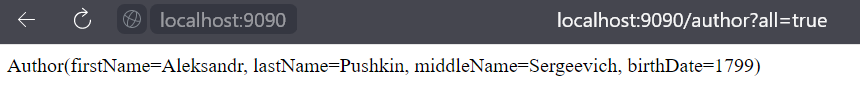
Main.java

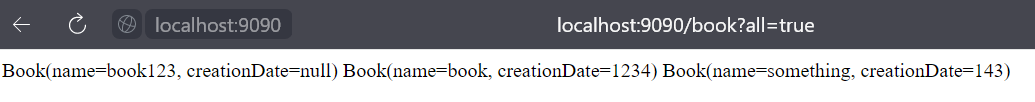
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class Main {  
 public static void main(String[] args) {  
 SpringApplication.*run*(Main.class, args);  
 }  
}

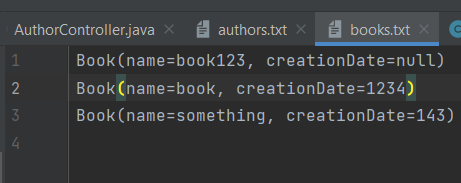
Student.html

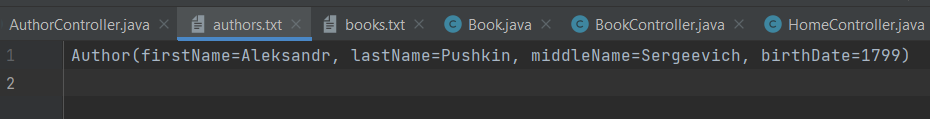
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
</head>  
<body>  
 <h1>Gegia Davit</h1>  
 <h2>IKBO-10-20</h2>  
 <h3>Variation: 3</h3>  
</body>  
</html>

1. **Результат работы программы**

****

****

****

****

1. **Выводы о проделанной работе**

В результате выполнения данной практической работы мы познакомились с принципами Spring MVC, начали работу с Rest API в Spring

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 и пояснения к нему**

Application.properties

spring.datasource.url=jdbc:postgresql://localhost:1234/javatasks  
hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect  
spring.datasource.username=postgres  
spring.datasource.password=123  
spring.jpa.generate-ddl=true  
entitymanager.packagesToScan: ru.mirea.fifteenth  
logging.file.name: application.log

Book.java

import com.fasterxml.jackson.annotation.JsonIgnore;  
import lombok.Getter;  
import lombok.Setter;  
import lombok.extern.slf4j.Slf4j;  
import ru.mirea.fifteenth.model.Author;  
  
import javax.persistence.\*;  
import javax.transaction.Transactional;  
  
@Slf4j  
@Getter  
@Setter  
@Entity  
@Table(name = "book")  
@Transactional  
public class Book {  
 @Id  
 @GeneratedValue( strategy =  
 GenerationType.*AUTO*)  
 private Long id;  
 @Column(name = "name")  
 private String name;  
 @Column(name = "date")  
 private String creationDate;  
  
 @JsonIgnore  
 @ManyToOne  
 public Author author;  
  
 public Book() {  
 }  
  
 public Book(String name, String creationDate, Author author) {  
 this.name = name;  
 this.creationDate = creationDate;  
 this.author = author;  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getCreationDate() {  
 return creationDate;  
 }  
  
 public void setCreationDate(String creationDate) {  
 this.creationDate = creationDate;  
 }  
  
 public Author getAuthor() {  
 return author;  
 }  
  
 @Override  
 public String toString() {  
 return "Book{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", creationDate='" + creationDate + '\'' +  
 '}';  
 }  
}

Author.java

import com.fasterxml.jackson.annotation.JsonIgnore;  
import lombok.Getter;  
import lombok.Setter;  
import lombok.ToString;  
import lombok.extern.slf4j.Slf4j;  
  
import javax.persistence.\*;  
import javax.transaction.Transactional;  
import java.util.List;  
  
@Slf4j  
@Entity  
@Table(name = "author")  
@Getter  
@Setter  
@ToString  
@Transactional  
public class Author {  
 @Id  
 @GeneratedValue( strategy =  
 GenerationType.*AUTO*)  
 private long id;  
  
 @Column(name = "firstName")  
 private String firstName;  
 @Column(name = "lastName")  
 private String lastName;  
 @Column(name = "middleName")  
 private String middleName;  
 @Column(name = "birthDate")  
 private String birthDate;  
  
 @JsonIgnore  
 @OneToMany(mappedBy = "author")  
 private List<Book> books;  
  
  
 public Author() {  
 }  
  
 public Author(String firstName, String lastName, String middleName, String birthDate) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.middleName = middleName;  
 this.birthDate = birthDate;  
 }  
  
 public List<Book> getBooks() {  
 return books;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
  
 public String getMiddleName() {  
 return middleName;  
 }  
  
 public void setMiddleName(String middleName) {  
 this.middleName = middleName;  
 }  
  
 public String getBirthDate() {  
 return birthDate;  
 }  
  
 public void setBirthDate(String birthDate) {  
 this.birthDate = birthDate;  
 }  
  
 public long getId() {  
 return id;  
 }  
}

AuthorController.java

import lombok.extern.slf4j.Slf4j;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
import ru.mirea.fifteenth.FifteenthApplication;  
import ru.mirea.fifteenth.model.Author;  
import ru.mirea.fifteenth.model.Book;  
import ru.mirea.fifteenth.repo.AuthorRepo;  
import ru.mirea.fifteenth.repo.BookRepo;  
import ru.mirea.fifteenth.service.EmailService;  
  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.List;  
  
@RestController  
@Slf4j  
public class AuthorController {  
 @Autowired  
 private AuthorRepo authorRepo;  
  
 @Autowired  
 private BookRepo bookRepo;  
  
 private static Logger *LOG* = LoggerFactory  
 .*getLogger*(FifteenthApplication.class);  
 @GetMapping("/author")  
 public @ResponseBody Iterable<Author> authorGet(@RequestParam(value = "firstName", defaultValue = "null") String name, @RequestParam(value = "all", defaultValue = "false") String all){  
 Iterable<Author> authors = authorRepo.findAll();  
 ArrayList<Author> list = new ArrayList<>();  
 try {  
 if(all.equals("true")){  
 for(Author k : authors){  
 list.add(k);  
 }  
 return list;  
 }  
 else {  
 for(Author k : authors){  
 if(k.getFirstName() == name){  
 list.add(k);  
 }  
 }  
 }  
 return list;  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
  
 @PostMapping("/author")  
 public @ResponseBody String authorPost(@RequestParam(value = "firstName", defaultValue = "null") String firstName, @RequestParam(value = "lastName", defaultValue = "null") String lastName, @RequestParam(value = "middleName", defaultValue = "null") String middleName, @RequestParam(value = "birthDate", defaultValue = "null") String birthDate) throws IOException {  
 Author author = new Author(firstName, lastName, middleName, birthDate);  
 authorRepo.save(author);  
 EmailService.*SendInfo*(author);  
 return "Success";  
 }  
  
 @DeleteMapping("/author")  
 public @ResponseBody String bookDel(@RequestParam(value = "name", defaultValue = "null") String name){  
 Iterable<Author> authors = authorRepo.findAll();  
 if(name == "null"){  
 return "null";  
 }  
  
 for(Author k : authors){  
 if(k.getFirstName().equals(name)){  
 authorRepo.deleteById(k.getId());  
 return "Success";  
 }  
 }  
 return "Fail";  
 }

}

BookController.java

import lombok.extern.slf4j.Slf4j;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.core.io.InputStreamResource;  
import org.springframework.core.io.Resource;  
import org.springframework.http.HttpHeaders;  
import org.springframework.http.MediaType;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import ru.mirea.fifteenth.FifteenthApplication;  
import ru.mirea.fifteenth.model.Author;  
import ru.mirea.fifteenth.model.Book;  
import ru.mirea.fifteenth.repo.AuthorRepo;  
import ru.mirea.fifteenth.repo.BookRepo;  
import ru.mirea.fifteenth.service.EmailService;  
  
import java.io.File;  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.List;  
  
@Slf4j  
@RestController  
public class BookController {  
 private static Logger *LOG* = LoggerFactory  
 .*getLogger*(FifteenthApplication.class);  
 @Autowired  
 private BookRepo bookRepo;  
 @Autowired  
 private AuthorRepo authorRepo;  
 @GetMapping("/book")  
 public @ResponseBody Iterable<Book> bookGet(@RequestParam(value = "name", defaultValue = "null") String name, @RequestParam(value = "all", defaultValue = "false") String all){  
  
 Iterable<Book> books = bookRepo.findAll();  
 ArrayList<Book> list = new ArrayList<>();  
 if(all.equals("true")){  
 for (Book k : books){  
 list.add(k);  
 }  
 return list;  
 }  
 else {  
 for (Book k : books){  
 if(k.getName().equals(name)){  
 list.add(bookRepo.findById(k.getId()).get());  
 }  
 }  
 return list;  
 }  
 }  
  
 @PostMapping("/book")  
 public @ResponseBody String bookPost(@RequestParam(value = "name", defaultValue = "null") String name, @RequestParam(value = "creationDate", defaultValue = "null") String creationDate, @RequestParam(value = "authorID", defaultValue = "1") String authorID) throws IOException {  
 Book book = new Book(name, creationDate, authorRepo.findById(Long.*parseLong*(authorID)).get());  
 bookRepo.save(book);  
 EmailService.*SendInfo*(book);  
 return "Success";  
 }  
  
 @DeleteMapping("/book")  
 public @ResponseBody String bookDel(@RequestParam(value = "name", defaultValue = "null") String name){  
 Iterable<Book> books = bookRepo.findAll();  
 for (Book k : books){  
 if(k.getName().equals(name)){  
 bookRepo.deleteById(k.getId());  
 return "Success";  
 }  
 }  
 return "Fail";  
 }

}

AuthorRepo.java

import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.CrudRepository;  
import org.springframework.stereotype.Repository;  
import ru.mirea.fifteenth.model.Author;  
  
import java.util.List;  
  
@Repository  
public interface AuthorRepo extends CrudRepository <Author, Long> {  
}

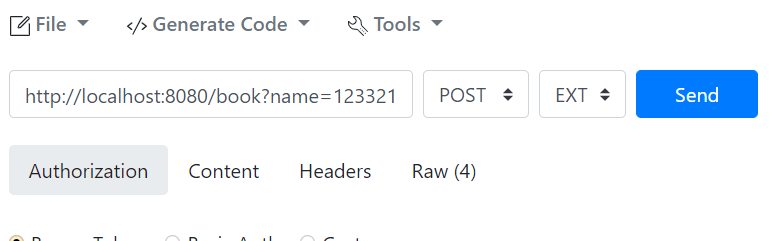
BookRepo.java

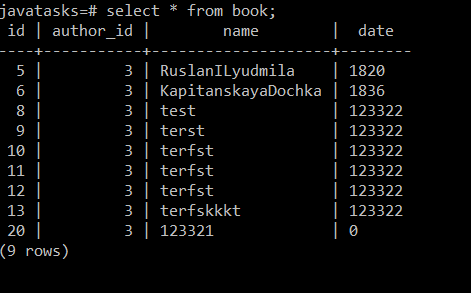
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.CrudRepository;  
import org.springframework.stereotype.Repository;  
import ru.mirea.fifteenth.model.Book;  
  
import java.util.List;  
  
@Repository  
public interface BookRepo extends CrudRepository <Book, Long> {  
  
}

FifteenthApplication.java

import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.scheduling.annotation.EnableScheduling;  
  
@SpringBootApplication  
@EnableScheduling  
public class FifteenthApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(FifteenthApplication.class, args);  
 }  
  
}

1. **Результат работы программы**





1. **Выводы о проделанной работе**

В результате выполнения данной практической работы мы познакомились с Hibernate и соединением spring web application с базой данных postgresql

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 и пояснения к нему**

Book.java

@JsonIgnore  
@ManyToOne  
public Author author;

Author.java

@JsonIgnore  
@OneToMany(mappedBy = "author")  
private List<Book> books;

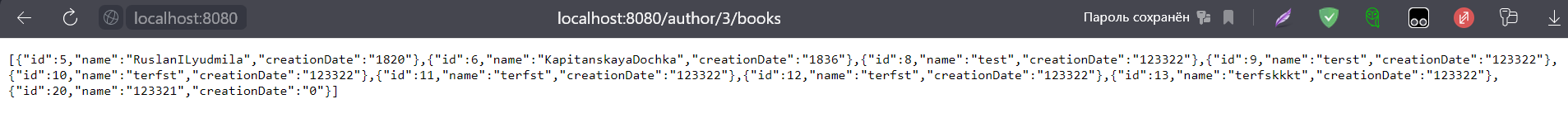
AuthorController.java

@GetMapping("/author/{author\_id}/books")  
public @ResponseBody Iterable<Book> getAuthorsBooks(@PathVariable("author\_id") String authorID){  
 List<Book> books = authorRepo.findById(Long.*parseLong*(authorID)).get().getBooks();  
 List<Long> list = new ArrayList<>();  
 for (Book book : books){  
 list.add(book.getId());  
 }  
 return bookRepo.findAllById(list);  
}

BookController.java

@GetMapping("book/{book\_id}/author")  
public @ResponseBody Author getBookAuthor(@PathVariable("book\_id") String book\_id){  
 return authorRepo.findById(bookRepo.findById(Long.*parseLong*(book\_id)).get().getAuthor().getId()).get();  
}

1. **Результат работы программы**

****

1. **Выводы о проделанной работе**

В результате выполнения данной практической работы мы научились создавать связи между сущностями в hibernate API

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 и пояснения к нему**

BookRepo.java

@Repository  
public interface BookRepo extends CrudRepository <Book, Long> {  
 @Query(value = "select \* from book order by name", nativeQuery = true)  
 List<Book> sortAllByName();  
 @Query(value = "select \* from book order by date", nativeQuery = true)  
 List<Book> sortAllByDate();  
 @Query(value = "select \* from book order by author\_id", nativeQuery = true)  
 List<Book> sortAllByAuthor();  
  
 @Query(value = "select \* from book", nativeQuery = true)  
 List<Book> getAllBooksByQuery();  
}

AuthorRepo.java

@Repository  
public interface AuthorRepo extends CrudRepository <Author, Long> {  
 @Query(value = "select \* from author order by first\_name", nativeQuery = true)  
 List<Author> sortAllByFirstName();  
 @Query(value = "select \* from author order by last\_name", nativeQuery = true)  
 List<Author> sortAllByLastName();  
 @Query(value = "select \* from author order by middle\_name", nativeQuery = true)  
 List<Author> sortAllByMiddleName();  
 @Query(value = "select \* from author order by birth\_date", nativeQuery = true)  
 List<Author> sortAllByBirthDate();  
  
 @Query(value = "select \* from author", nativeQuery = true)  
 List<Author> getAllAuthorsByQuery();  
}

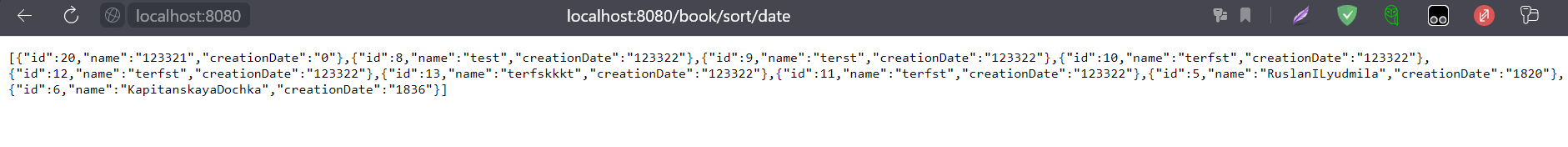
BookController.java

@GetMapping("book/sort/name")  
public @ResponseBody Iterable<Book> bookGetFilterByName(){  
 List<Book> list = bookRepo.sortAllByName();  
 return list;  
}  
  
@GetMapping("book/sort/date")  
public @ResponseBody Iterable<Book> bookGetFilterByDate(){  
 List<Book> list = bookRepo.sortAllByDate();  
 return list;  
}  
  
@GetMapping("book/sort/author")  
public @ResponseBody Iterable<Book> bookGetFilterByAuthor(){  
 List<Book> list = bookRepo.sortAllByAuthor();  
 return list;  
}

AuthorController.java

@GetMapping("author/sort/firstName")  
public @ResponseBody Iterable<Author> authorGetFilterByFirstName(){  
 List<Author> list = authorRepo.sortAllByFirstName();  
 return list;  
}  
  
@GetMapping("author/sort/lastName")  
public @ResponseBody Iterable<Author> authorGetFilterByLastName(){  
 List<Author> list = authorRepo.sortAllByLastName();  
 return list;  
}  
  
@GetMapping("author/sort/middleName")  
public @ResponseBody Iterable<Author> authorGetFilterByMiddleName(){  
 List<Author> list = authorRepo.sortAllByMiddleName();  
 return list;  
}  
  
@GetMapping("author/sort/birthDate")  
public @ResponseBody Iterable<Author> authorGetFilterByBirthDate(){  
 List<Author> list = authorRepo.sortAllByBirthDate();  
 return list;  
}

1. **Результат работы программы**



1. **Выводы о практической работе**

В результате выполнения данной практической работы мы познакомились с Criteria API и научились фильтровать и сортировать дынные из базы данных

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 и пояснения к нему**

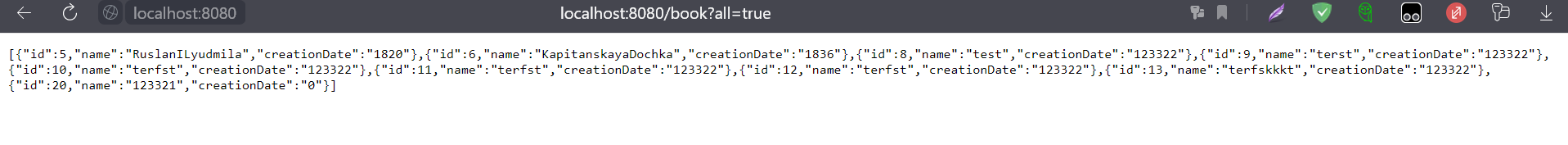
BookRepo.java

import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.CrudRepository;  
import org.springframework.stereotype.Repository;  
import ru.mirea.fifteenth.model.Book;  
  
import java.util.List;  
  
@Repository  
public interface BookRepo extends CrudRepository <Book, Long> {  
 @Query(value = "select \* from book order by name", nativeQuery = true)  
 List<Book> sortAllByName();  
 @Query(value = "select \* from book order by date", nativeQuery = true)  
 List<Book> sortAllByDate();  
 @Query(value = "select \* from book order by author\_id", nativeQuery = true)  
 List<Book> sortAllByAuthor();  
  
 @Query(value = "select \* from book", nativeQuery = true)  
 List<Book> getAllBooksByQuery();  
}

AuthorRepo.java

import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.CrudRepository;  
import org.springframework.stereotype.Repository;  
import ru.mirea.fifteenth.model.Author;  
  
import java.util.List;  
  
@Repository  
public interface AuthorRepo extends CrudRepository <Author, Long> {  
 @Query(value = "select \* from author order by first\_name", nativeQuery = true)  
 List<Author> sortAllByFirstName();  
 @Query(value = "select \* from author order by last\_name", nativeQuery = true)  
 List<Author> sortAllByLastName();  
 @Query(value = "select \* from author order by middle\_name", nativeQuery = true)  
 List<Author> sortAllByMiddleName();  
 @Query(value = "select \* from author order by birth\_date", nativeQuery = true)  
 List<Author> sortAllByBirthDate();  
  
 @Query(value = "select \* from author", nativeQuery = true)  
 List<Author> getAllAuthorsByQuery();  
}

1. **Результат работы программы**

****

1. **Выводы о практической работе**

В результате выполнения данной практической работы мы познакомились с Spring data JPA и переписали программу под неё

Практическая работа №19

1. **Постановка задачи и персональный вариант**

Тема: Знакомство с логированием с использованием Logback в Spring.

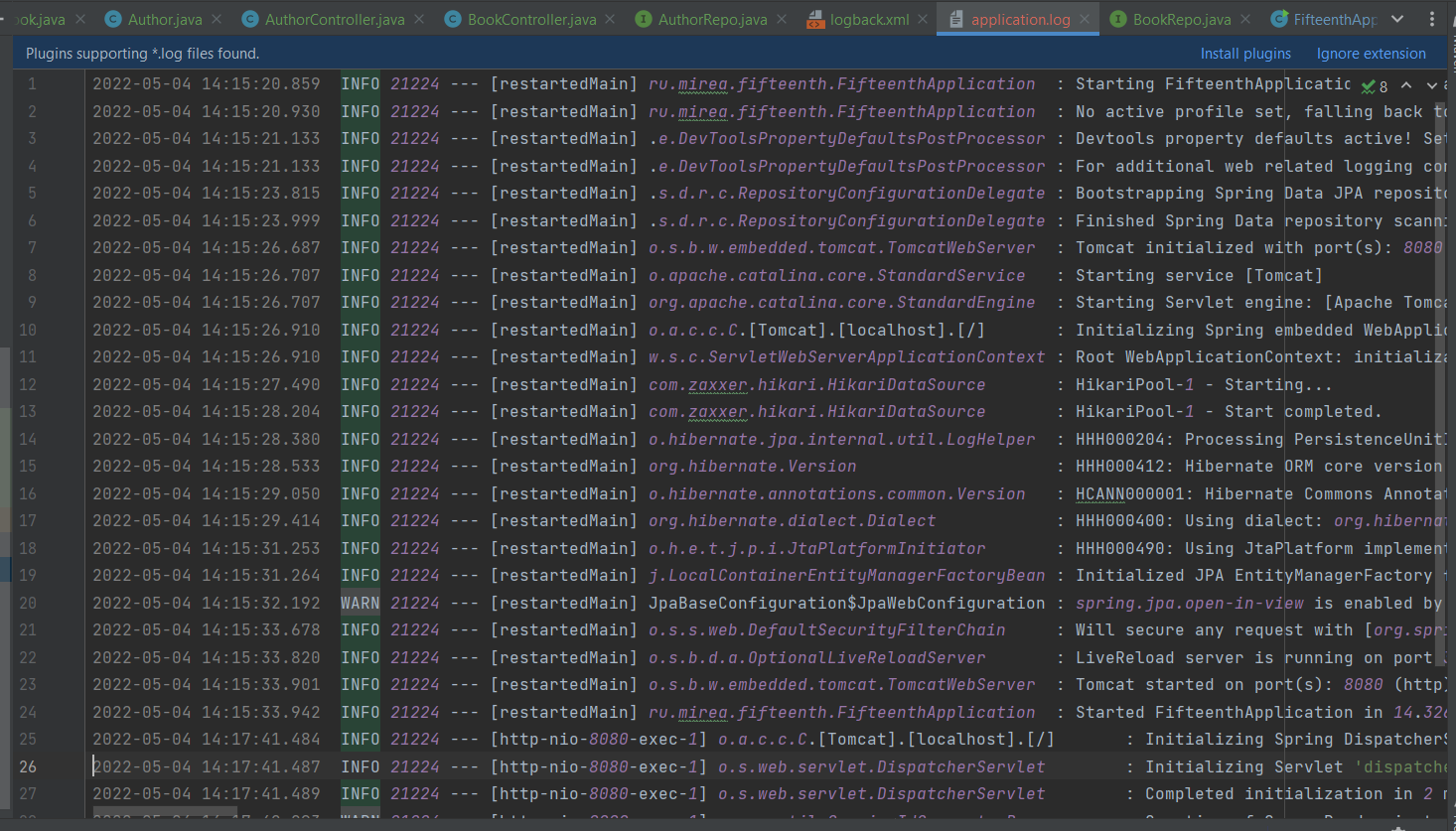
Постановка задачи: Создать файл logback.xml, добавить логирование во все методы классов-сервисов.

1. **Код и пояснения к нему**

Logback.xml

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

1. **Результат работы программы**



1. **Выводы о практической работе**

В результат выполнения данной практической работы мы познакомились с принципами логгирования в Spring boot приложения, а также добавили его во все методы.

Практическая работа №20

1. **Постановка задачи и персональный вариант**

Тема: Использование Spring AOP. Pointcut, JoinPoint. Advice.

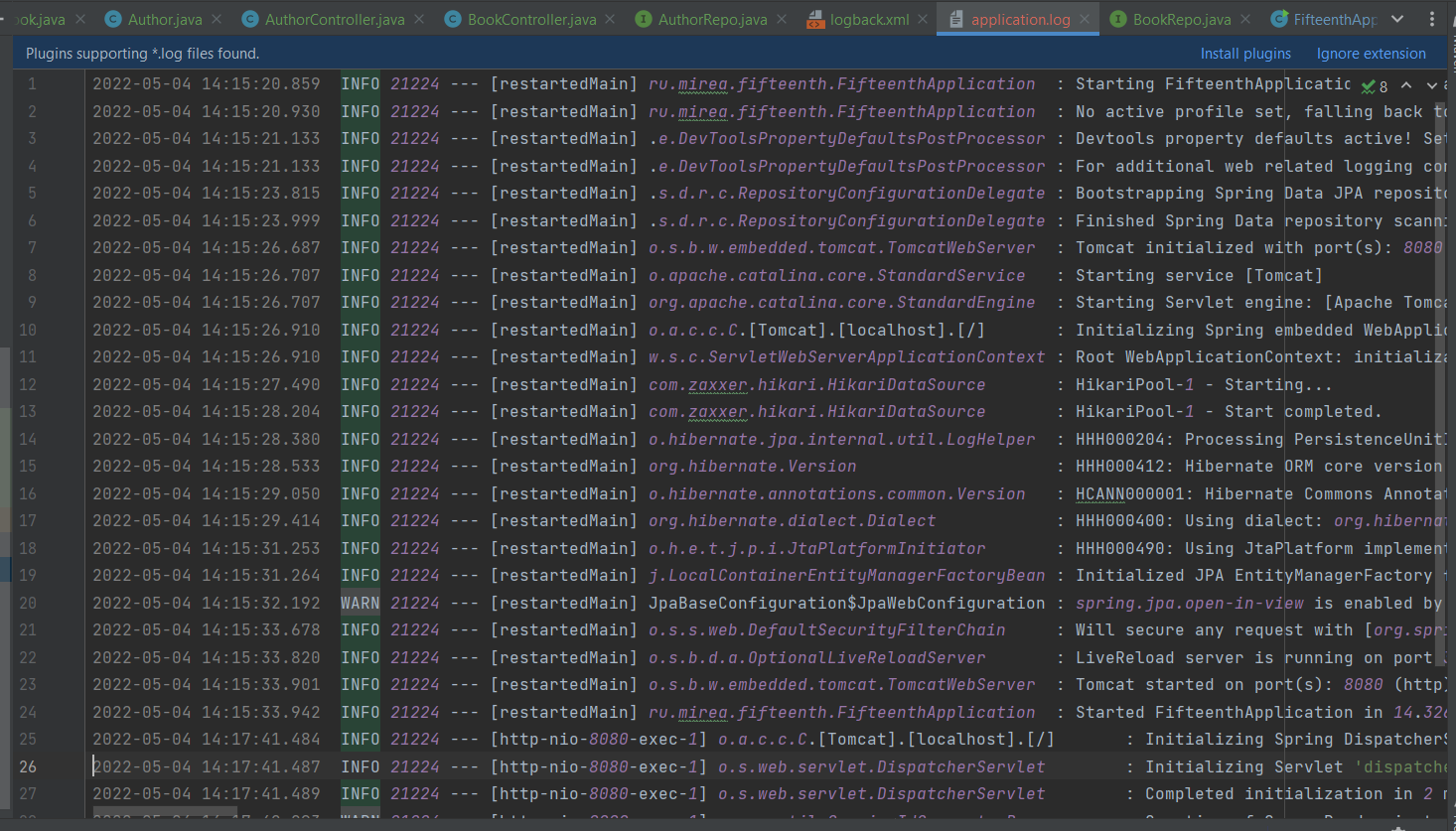
Постановка задачи: Для приложения из предыдущего задания добавить логирование времени выполнения каждого метода сервиса с использованием Spring AOP

1. **Код и пояснения к нему**

Aspect.java

import lombok.extern.slf4j.Slf4j;  
import org.aspectj.lang.JoinPoint;  
import org.aspectj.lang.annotation.Before;  
import org.aspectj.lang.annotation.Pointcut;  
import org.springframework.stereotype.Component;  
  
@org.aspectj.lang.annotation.Aspect  
@Slf4j  
@Component  
public class Aspect {  
 @Before("allServiceMethods()")  
 public void logParameters(JoinPoint joinPoint) {  
 *log*.info("Parameters: {}", joinPoint.getArgs());  
 }  
 @Pointcut("within(ru.mirea.fifteenth.\*)")  
 public void allServiceMethods() {}  
}

1. **Результат работы программы**



1. **Выводы о практической работе**

В результате выполнения данной практической работы мы познакомились с Spring AOP принципами, аннотациями Aspect, Joinpoint и Advice

Практическая работа № 21

1. **Постановка задачи и персональный вариант**

Тема: Проксирование. Аннотация Transactional. Аннотация Async.

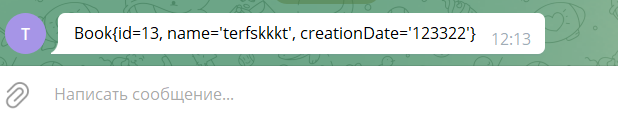
Постановка задачи: Для приложения из предыдущего задания пометить все классы сервисов, в которых происходит взаимодействие с базой данных, как Transactional. Добавить отправку информации о сохранении каждого объекта по электронной почте, создав отдельный класс EmailService с асинхронными методами отправки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

1. **Код и пояснения к нему**

EmailService.java

import lombok.extern.slf4j.Slf4j;  
import org.apache.http.NameValuePair;  
import org.apache.http.client.entity.UrlEncodedFormEntity;  
import org.apache.http.client.methods.CloseableHttpResponse;  
import org.apache.http.client.methods.HttpPost;  
import org.apache.http.impl.client.CloseableHttpClient;  
import org.apache.http.impl.client.HttpClients;  
import org.apache.http.message.BasicNameValuePair;  
import org.apache.http.util.EntityUtils;  
import org.springframework.scheduling.annotation.Async;  
import org.springframework.stereotype.Component;  
import ru.mirea.fifteenth.model.Author;  
import ru.mirea.fifteenth.model.Book;  
  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.List;  
  
@Component  
@Slf4j  
@Async  
public class EmailService {  
 public static void SendInfo(Book book) throws IOException {  
 HttpPost httpPost = new HttpPost("https://api.telegram.org/bot5151738188:AAHs14q0d5vS38bOOOEIORwrWiPDvA3bJ4E/sendMessage");  
 List<NameValuePair> urlParameters = new ArrayList<>();  
 urlParameters.add(new BasicNameValuePair("chat\_id", "336430443"));  
 urlParameters.add(new BasicNameValuePair("text", book.toString()));  
 httpPost.setEntity(new UrlEncodedFormEntity(urlParameters));  
  
 try (CloseableHttpClient httpClient = HttpClients.*createDefault*();  
 CloseableHttpResponse response = httpClient.execute(httpPost)) {  
  
 *log*.info(EntityUtils.*toString*(response.getEntity()));  
 }  
  
 }  
 public static void SendInfo(Author author) throws IOException {  
 HttpPost httpPost = new HttpPost("https://api.telegram.org/bot5151738188:AAHs14q0d5vS38bOOOEIORwrWiPDvA3bJ4E/sendMessage");  
 List<NameValuePair> urlParameters = new ArrayList<>();  
 urlParameters.add(new BasicNameValuePair("chat\_id", "336430443"));  
 urlParameters.add(new BasicNameValuePair("text", author.toString()));  
 httpPost.setEntity(new UrlEncodedFormEntity(urlParameters));  
  
 try (CloseableHttpClient httpClient = HttpClients.*createDefault*();  
 CloseableHttpResponse response = httpClient.execute(httpPost)) {  
  
 *log*.info(EntityUtils.*toString*(response.getEntity()));  
 }  
 }  
}

1. **Результат работы программы**

****

1. **Выводы о практической работе**

В результате выполнения данной практической работы мы познакомились с аннотациями Tranactional и Async, реализовали сервис по пересылке сообщений.

Практическая работа №22

1. **Постановка задачи и персональный вариант**

Тема: Планирование заданий. Scheduler в Spring

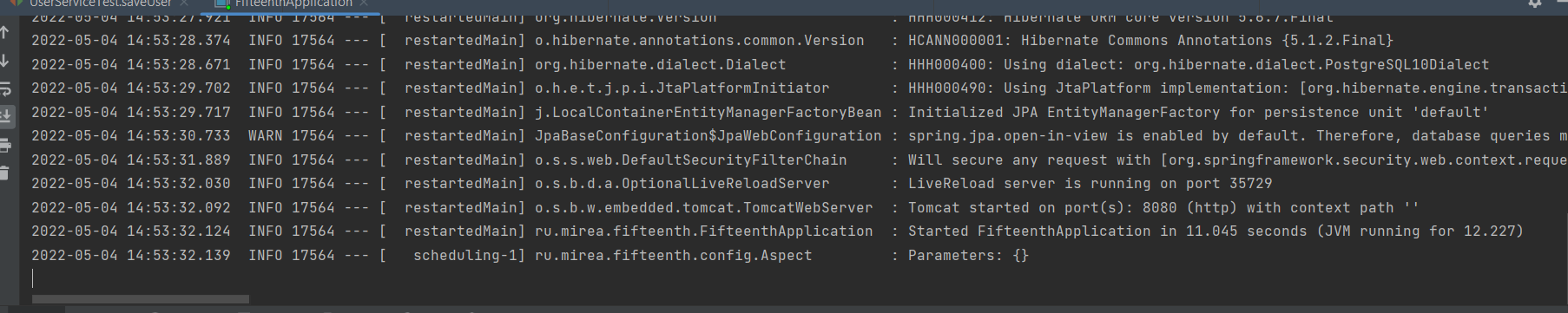
Постановка задачи: Для приложения из предыдущего задания создать класс-сервис с методом, который будет вызываться каждые 30 минут и очищать определённую директорию, а затем создавать по файлу для каждой из сущностей и загружать туда все данные из базы данных. Также добавить возможность вызывать данный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

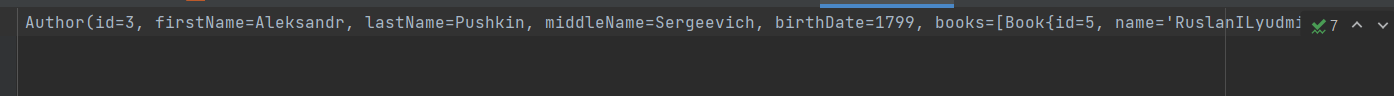
1. **Код и пояснения к нему**

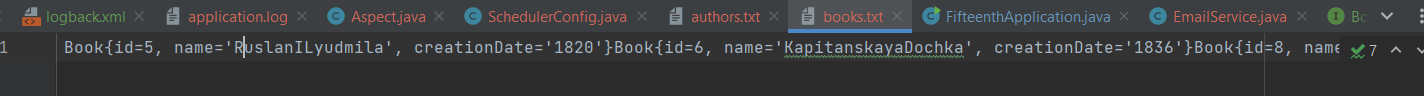
SchedulerConfig.java

import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.scheduling.annotation.Scheduled;  
import org.springframework.stereotype.Component;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import ru.mirea.fifteenth.model.Author;  
import ru.mirea.fifteenth.repo.AuthorRepo;  
import ru.mirea.fifteenth.model.Book;  
import ru.mirea.fifteenth.repo.BookRepo;  
  
import java.io.File;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.util.List;  
  
@Service  
@Component  
@Transactional  
public class SchedulerConfig {  
 @Autowired  
 private BookRepo bookRepo;  
 @Autowired  
 private AuthorRepo authorRepo;  
 @Scheduled(fixedDelayString = "PT30S")  
 public void doMagic() throws IOException {  
 File dir = new File("C:\\Users\\User\\Desktop\\fifteenth\\src\\main\\java\\ru\\mirea\\fifteenth\\dir");  
 String[] entries = dir.list();  
 for (String str : entries){  
 File currentFile = new File(dir.getPath(), str);  
 currentFile.delete();  
 }  
 File books = new File(dir.getPath(), "books.txt");  
 File authors = new File(dir.getPath(), "authors.txt");  
 books.createNewFile();  
 authors.createNewFile();  
 FileWriter writer1 = new FileWriter(books);  
 FileWriter writer2 = new FileWriter(authors);  
 List<Book> bookList = bookRepo.getAllBooksByQuery();  
 List<Author> authorList = authorRepo.getAllAuthorsByQuery();  
 for (Book book : bookList){  
 writer1.write(book.toString());  
 writer1.flush();  
 }  
 for (Author author : authorList){  
 writer2.write(author.toString());  
 writer2.flush();  
 }  
 }  
}

1. **Результат работы программы**







1. **Выводы о практической работе**

В результате выполнения данной практической работы мы познакомились с планированием заданий в Spring.

Практическая работа № 23

1. **Постановка задачи и персональный вариант**

Тема: Использование Spring Security для аутентификации и авторизации пользователей.Постановка задачи: В приложении из предыдущего задания добавить возможность регистрации и авторизации пользователей, хранение cookie сессий в базе данных PostgreSQL, хеширование паролей алгоритмом Bcrypt, защиту всех запросов, кроме запросов на авторизацию и регистрацию, от неавторизированных пользователей.

1. **Код и пояснения к нему**

SecurityConfig.java

import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.context.annotation.Lazy;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.config.BeanIds;  
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
import org.springframework.security.config.http.SessionCreationPolicy;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
  
@Configuration  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Lazy  
 @Autowired  
 private UserDetailsService userDetailsService;  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http.csrf().disable().cors().disable()  
 .authorizeRequests()  
 .antMatchers("/login", "/register").permitAll()  
 .anyRequest().authenticated()  
 .and().formLogin()  
 .and().userDetailsService(userDetailsService());  
 http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.*ALWAYS*);;  
 }  
  
 @Override  
 protected void configure(AuthenticationManagerBuilder auth) throws Exception {  
 super.configure(auth);  
 auth  
 .userDetailsService(userDetailsService);  
 }  
  
 @Bean  
 public BCryptPasswordEncoder passwordEncoder() {  
 BCryptPasswordEncoder bCryptPasswordEncoder = new BCryptPasswordEncoder();  
 return bCryptPasswordEncoder;  
 }  
  
 @Bean(name = BeanIds.*AUTHENTICATION\_MANAGER*)  
 @Override  
 public AuthenticationManager authenticationManagerBean() throws Exception {  
 return super.authenticationManagerBean();  
 }  
  
 @Bean  
 public UserDetailsService userDetailsService() {  
 return super.userDetailsService();  
 }  
}

User.java

import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
import lombok.extern.slf4j.Slf4j;  
  
import javax.persistence.\*;  
import javax.transaction.Transactional;  
  
@Slf4j  
@Getter  
@Setter  
@Entity  
@Table(name = "users")  
@Transactional  
@NoArgsConstructor  
public class User {  
 @Id  
 @GeneratedValue( strategy =  
 GenerationType.*AUTO*)  
 private Long id;  
  
 @Column(name = "username")  
 private String username;  
  
 @Column(name = "hashpass")  
 private String hashPass;  
  
 public User(String username, String hashPass) {  
 this.username = username;  
 this.hashPass = hashPass;  
 }  
}

UserController.java

import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.web.bind.annotation.\*;  
import ru.mirea.fifteenth.model.User;  
  
@RestController  
public class UserController {  
 @Autowired  
 UserRepo userRepo;  
  
 @Autowired  
 BCryptPasswordEncoder bCryptPasswordEncoder;  
  
 @PostMapping("/register")  
 public String register(@RequestParam(value = "username", defaultValue = "null") String username, @RequestParam(value = "password", defaultValue = "null") String password){  
 if(username.equals("null") || password.equals("null")){  
 return null;  
 }  
 userRepo.save(new User(username, bCryptPasswordEncoder.encode(password)));  
  
 return "redirect:/home";  
 }  
  
  
  
}

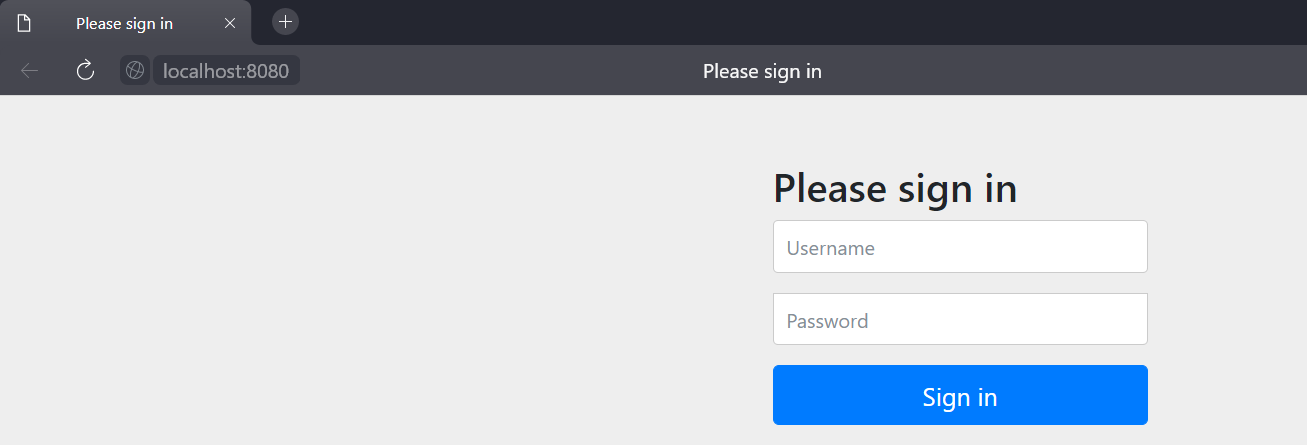
UserRepo.java

import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.CrudRepository;  
import org.springframework.stereotype.Repository;  
import ru.mirea.fifteenth.model.User;  
  
@Repository  
public interface UserRepo extends CrudRepository<User, Long> {  
 @Query(value = "select \* from users where username = ?1", nativeQuery = true)  
 User findUserByUsername(String username);  
}

UserService.java

import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.stereotype.Service;  
import ru.mirea.fifteenth.model.User;  
import ru.mirea.fifteenth.repo.UserRepo;  
  
import java.util.List;  
  
@Service  
public class UserService implements UserDetailsService {  
  
 @Autowired  
 UserRepo userRepo;  
 @Autowired  
 private BCryptPasswordEncoder bCryptPasswordEncoder;  
  
 public void saveUser(User user) {  
 userRepo.save(new User(user.getUsername(), bCryptPasswordEncoder.encode(user.getHashPass())));  
 }  
  
 public User findByUsername(String username) {  
 return userRepo.findUserByUsername(username);  
 }  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 User user = userRepo.findUserByUsername(username);  
 return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getHashPass(), List.*of*());  
 }  
}

1. **Результат работы программы**

****

1. **Выводы о практической работе**

В результате выполнения данной практической работы мы познакомились с Spring security для аутентификации пользователей.

Практическая работа №24

1. **Постановка задачи и персональный вариант**

Тема: Тестирование в Spring Framework с использованием Junit.Постановка задачи: Написать модульное тестирование для всех классов сервисов приложения из предыдущего задания.

1. **Код и комментарии к нему**

UserServiceTest.java

import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.mockito.ArgumentCaptor;  
import org.mockito.Captor;  
import org.mockito.Mock;  
import org.mockito.junit.jupiter.MockitoExtension;  
import ru.mirea.fifteenth.model.User;  
import ru.mirea.fifteenth.repo.UserRepo;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.mockito.ArgumentMatchers.*any*;  
import static org.mockito.Mockito.*verify*;  
import static org.mockito.Mockito.*when*;  
  
@ExtendWith(MockitoExtension.class)  
public class UserServiceTest {  
 @Mock  
 private UserRepo userRepo;  
  
 @Captor  
 ArgumentCaptor<User> captor;  
  
 @Test  
 void saveUser(){  
 User user = new User("Vasya", "test");  
 *when*(userRepo.save(*any*())).thenReturn(user);  
 User newUser = userRepo.save(user);  
 *verify*(userRepo).save(captor.capture());  
 User newnewUser = captor.getValue();  
 *assertEquals*(user.getUsername(), newUser.getUsername());  
 *assertEquals*(user.getHashPass(), newnewUser.getHashPass());  
 }  
  
 @Test  
 void findByUsername(){  
 User user = new User("Test", "Test");  
 *when*(userRepo.findUserByUsername(*any*())).thenReturn(user);  
 User newUser = userRepo.findUserByUsername("Test");  
 *assertEquals*(user, newUser);  
 }  
}

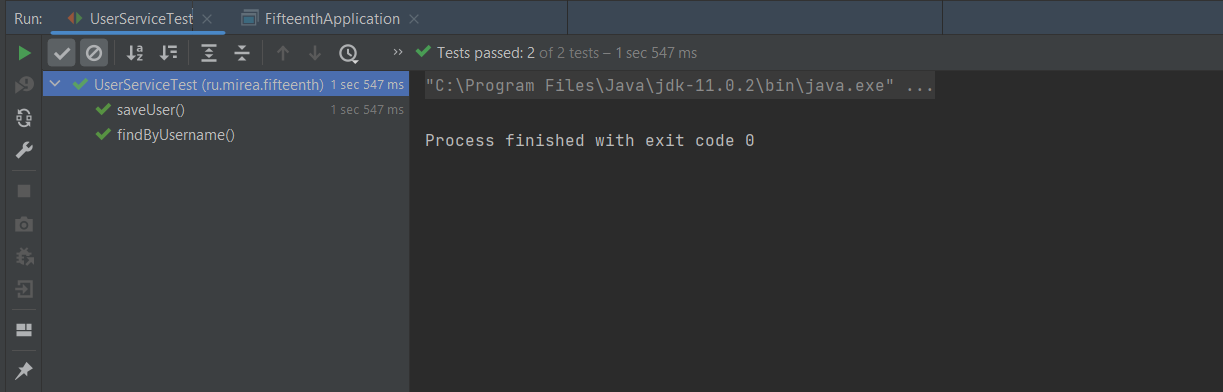
AuthorControllerTest.java

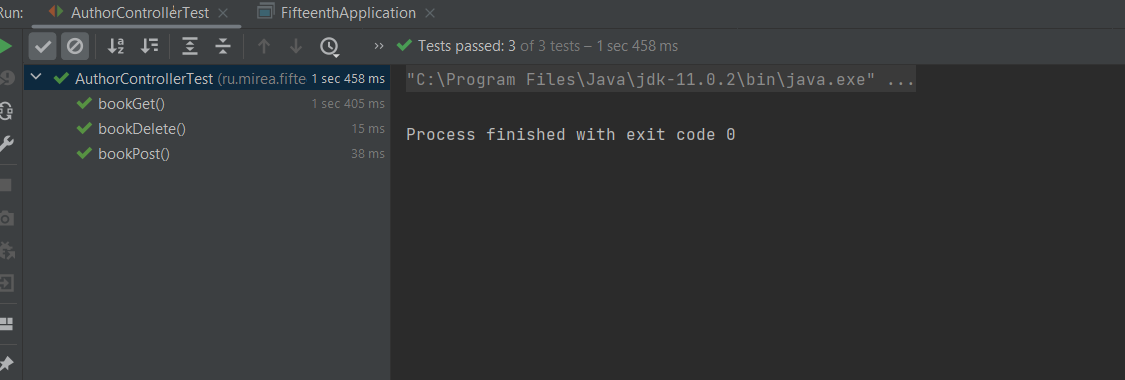
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.mockito.ArgumentCaptor;  
import org.mockito.Captor;  
import org.mockito.Mock;  
import org.mockito.junit.jupiter.MockitoExtension;  
import ru.mirea.fifteenth.model.Author;  
import ru.mirea.fifteenth.model.Author;  
import ru.mirea.fifteenth.repo.AuthorRepo;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.mockito.ArgumentMatchers.*any*;  
import static org.mockito.Mockito.*verify*;  
import static org.mockito.Mockito.*when*;  
  
@ExtendWith(MockitoExtension.class)  
public class AuthorControllerTest {  
 @Mock  
 AuthorRepo authorRepo;  
  
 @Captor  
 ArgumentCaptor<Author> captor;  
  
 @Test  
 void bookGet(){  
 Author author = new Author("testname", "testlastname", "testmiddlename", "0");  
 List<Author> authors = new ArrayList<>();  
 authors.add(author);  
 *when*(authorRepo.findAll()).thenReturn(authors);  
 Iterable<Author> newAuthors = authorRepo.findAll();  
 Author neeededAuthor = new Author();  
 for (Author itAuthor : newAuthors){  
 if(itAuthor.getId() == author.getId()){  
 neeededAuthor = itAuthor;  
 }  
 }  
 *assertEquals*(author.getId(), neeededAuthor.getId());  
 }  
  
 @Test  
 void bookPost(){  
 Author author = new Author("testname", "testlastname", "testmiddlename", "0");  
 *when*(authorRepo.save(*any*())).thenReturn(author);  
 Author newBook = authorRepo.save(author);  
 *verify*(authorRepo).save(captor.capture());  
 Author newnewBook = captor.getValue();  
 *assertEquals*(author.getId(), newBook.getId());  
 *assertEquals*(author.getId(), newnewBook.getId());  
 }  
  
 @Test  
 void bookDelete(){  
 Author author = new Author("testname", "testlastname", "testmiddlename", "0");  
 Author author2 = new Author("testname", "testlastname", "testmiddlename", "0");  
 List<Author> authors = new ArrayList<>();  
 List<Author> authors1 = new ArrayList<>();  
 List<Author> authors2 = new ArrayList<>();  
 authors.add(author);  
 authors1.add(author);  
 authors2.add(author);  
 authors.add(author2);  
 authors2.add(author2);  
 *when*(authorRepo.findAll()).thenReturn(authors);  
 Iterable<Author> newBooks = authorRepo.findAll();  
 for (Author itBook : newBooks){  
 if(itBook == author){  
 authors2.remove(itBook);  
 }  
 }  
 *assertEquals*(authors2.get(0).getId(), authors1.get(0).getId());  
 }  
}

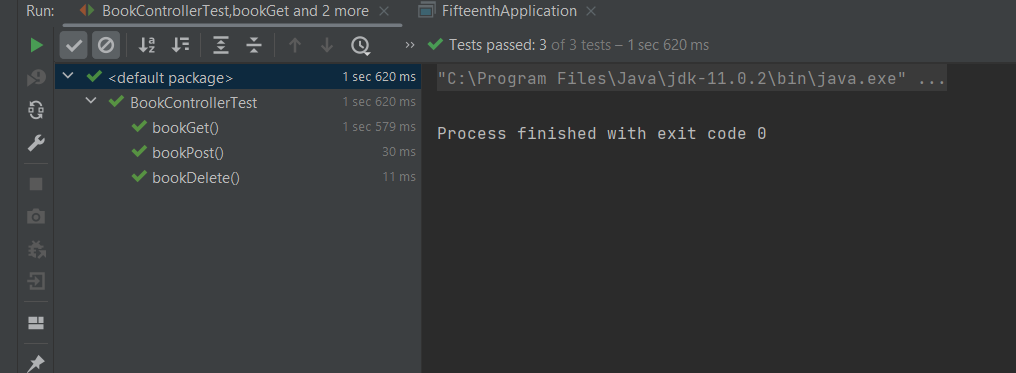
UserServiceTest.java

import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.mockito.ArgumentCaptor;  
import org.mockito.Captor;  
import org.mockito.Mock;  
import org.mockito.junit.jupiter.MockitoExtension;  
import ru.mirea.fifteenth.model.Author;  
import ru.mirea.fifteenth.model.Author;  
import ru.mirea.fifteenth.repo.AuthorRepo;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.mockito.ArgumentMatchers.*any*;  
import static org.mockito.Mockito.*verify*;  
import static org.mockito.Mockito.*when*;  
  
@ExtendWith(MockitoExtension.class)  
public class AuthorControllerTest {  
 @Mock  
 AuthorRepo authorRepo;  
  
 @Captor  
 ArgumentCaptor<Author> captor;  
  
 @Test  
 void bookGet(){  
 Author author = new Author("testname", "testlastname", "testmiddlename", "0");  
 List<Author> authors = new ArrayList<>();  
 authors.add(author);  
 *when*(authorRepo.findAll()).thenReturn(authors);  
 Iterable<Author> newAuthors = authorRepo.findAll();  
 Author neeededAuthor = new Author();  
 for (Author itAuthor : newAuthors){  
 if(itAuthor.getId() == author.getId()){  
 neeededAuthor = itAuthor;  
 }  
 }  
 *assertEquals*(author.getId(), neeededAuthor.getId());  
 }  
  
 @Test  
 void bookPost(){  
 Author author = new Author("testname", "testlastname", "testmiddlename", "0");  
 *when*(authorRepo.save(*any*())).thenReturn(author);  
 Author newBook = authorRepo.save(author);  
 *verify*(authorRepo).save(captor.capture());  
 Author newnewBook = captor.getValue();  
 *assertEquals*(author.getId(), newBook.getId());  
 *assertEquals*(author.getId(), newnewBook.getId());  
 }  
  
 @Test  
 void bookDelete(){  
 Author author = new Author("testname", "testlastname", "testmiddlename", "0");  
 Author author2 = new Author("testname", "testlastname", "testmiddlename", "0");  
 List<Author> authors = new ArrayList<>();  
 List<Author> authors1 = new ArrayList<>();  
 List<Author> authors2 = new ArrayList<>();  
 authors.add(author);  
 authors1.add(author);  
 authors2.add(author);  
 authors.add(author2);  
 authors2.add(author2);  
 *when*(authorRepo.findAll()).thenReturn(authors);  
 Iterable<Author> newBooks = authorRepo.findAll();  
 for (Author itBook : newBooks){  
 if(itBook == author){  
 authors2.remove(itBook);  
 }  
 }  
 *assertEquals*(authors2.get(0).getId(), authors1.get(0).getId());  
 }  
}

1. **Результат работы программы**







1. **Выводы о практической работе**

В результате выполнения практической работы мы познакомились с библиотекой Mockito и провели модульное тестирование всех методов контроллеров.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Методические указания к выполнению практических работ [Электронный ресурс] // URL: <https://online-edu.mirea.ru/pluginfile.php?file=%2F697655%2Fmod_resource%2Fcontent%2F2%2FМетодические%20указания.pdf>
2. Конспект лекций Зорина Н.В. [Электронный ресурс] // URL: <https://online-edu.mirea.ru/mod/folder/view.php?id=372300>
3. Стелтинг С., Маасен О. Применение шаблонов Java. Библиотека профессионала.: Пер. с англ. — М.: Издательский дом "Вильяме", 2002. — 576 с.: ил. — Парал. тит. Англ
4. Hibernate Search 6.0.0.Final: Reference Documentation [Электронный ресурс]. URL: <https://docs.jboss.org/hibernate/stable/search/reference/enUS/html_single/>
5. Spring Framework Documentation [Электронный ресурс]. URL: <https://docs.spring.io/spring-framework/docs/current/reference/html/web.html>