|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА − Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий (ИИТ)**

**Кафедра инструментального и прикладного программного обеспечения (ИиППО)**

**ОТЧЁТ ПО ПРАКТИЧЕСКИМ РАБОТАМ №1-24**

**по дисциплине «Шаблоны программных платформ языка Джава»**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил:  Студент группы ИКБО-20-22 | « 27 » мая 2024г. | | (подпись) | | Фридрих А.В. | |
|  | |  | |  | |  | |
| Принял:  Преподаватель кафедры ИиППО ИИТ | | « 27 » мая 2024г. | | (подпись) | | Ермаков С.Р. | |

Москва 2023 г.

**Содержание**

[1 Практическая работа №1 7](#_Toc152328307)

[1.1 Цель работы. 7](#_Toc152328308)

[1.2 Задание. 7](#_Toc152328309)

[1.3 Тестирование. 9](#_Toc152328310)

[1.4 Выводы. 9](#_Toc152328311)

[2 Практическая работа №2 10](#_Toc152328312)

[2.1 Цель работы. 10](#_Toc152328313)

[2.2 Задание. 10](#_Toc152328314)

[2.3 Код программы. 10](#_Toc152328315)

[2.4 Тестирование. 12](#_Toc152328316)

[2.5 Вывод. 12](#_Toc152328317)

[3 Практическая работа №3 13](#_Toc152328318)

[3.1 Цель работы. 13](#_Toc152328319)

[3.2 Задание. 13](#_Toc152328320)

[3.3 Код программы. 13](#_Toc152328321)

[3.4 Тестирование. 14](#_Toc152328322)

[3.5 Вывод. 15](#_Toc152328323)

[4 Практическая работа №4 16](#_Toc152328324)

[4.1 Цель работы. 16](#_Toc152328325)

[4.2 Задание. 16](#_Toc152328326)

[4.3 Код программы. 16](#_Toc152328327)

[4.4 Тестирование. 19](#_Toc152328328)

[4.5 Вывод. 19](#_Toc152328329)

[5 Практическая работа №5 20](#_Toc152328330)

[5.1 Цель работы. 20](#_Toc152328331)

[5.2 Задание. 20](#_Toc152328332)

[5.3 Код программы. 22](#_Toc152328333)

[5.4 Тестирование. 36](#_Toc152328334)

[5.5 Вывод. 36](#_Toc152328335)

[6 Практическая работа №6 37](#_Toc152328336)

[6.1 Цель работы. 37](#_Toc152328337)

[6.2 Задание. 37](#_Toc152328338)

[6.3 Код программы. 37](#_Toc152328339)

[6.4 Тестирование. 41](#_Toc152328340)

[6.5 Вывод. 41](#_Toc152328341)

[7 Практическая работа №7 42](#_Toc152328342)

[7.1 Цель. 42](#_Toc152328343)

[7.2 Задание. 42](#_Toc152328344)

[7.3 Код программы. 42](#_Toc152328345)

[7.4 Тестирование. 44](#_Toc152328346)

[7.5 Вывод. 44](#_Toc152328347)

[8 Практическая работа №8 45](#_Toc152328348)

[8.1 Цель работы. 45](#_Toc152328349)

[8.2 Задание. 45](#_Toc152328350)

[8.3 Код программы. 45](#_Toc152328351)

[8.4 Тестирование. 51](#_Toc152328352)

[8.5 Вывод. 51](#_Toc152328353)

[9 Практические работы №9 52](#_Toc152328354)

[9.1 Цель работы. 52](#_Toc152328355)

[9.2 Задание. 52](#_Toc152328356)

[9.3 Тестирование. 54](#_Toc152328357)

[9.4 Вывод. 54](#_Toc152328358)

[10 Практическая работа №10 55](#_Toc152328359)

[10.1 Цель работы. 55](#_Toc152328360)

[10.2 Задание. 55](#_Toc152328361)

[10.3 Код программы. 55](#_Toc152328362)

[10.4 Тестирование. 58](#_Toc152328363)

[10.5 Вывод. 58](#_Toc152328364)

[11 Практическое задание №11 59](#_Toc152328365)

[11.1 Цель работы. 59](#_Toc152328366)

[11.2 Задание. 59](#_Toc152328367)

[11.3 Код программы. 59](#_Toc152328368)

[11.4 Тестиривание. 66](#_Toc152328369)

[11.5 Вывод. 67](#_Toc152328370)

[12 Практическая работа №12 68](#_Toc152328371)

[12.1 Цель работы. 68](#_Toc152328372)

[12.2 Задание. 68](#_Toc152328373)

[12.3 Код программы. 68](#_Toc152328374)

[12.4 Тестирование. 75](#_Toc152328375)

[12.5 Вывод. 75](#_Toc152328376)

[13 Практическая работа №13 76](#_Toc152328377)

[13.1 Цель работы. 76](#_Toc152328378)

[13.2 Задание. 77](#_Toc152328379)

[13.3 Код программы. 77](#_Toc152328380)

[13.4 Тестирование. 77](#_Toc152328381)

[13.5 Вывод. 78](#_Toc152328382)

[14 Практическая работа №14 78](#_Toc152328383)

[14.1 Цель работы. 78](#_Toc152328384)

[14.2 Задание. 78](#_Toc152328385)

[14.3 Код программы. 79](#_Toc152328386)

[14.4 Тестирование. 86](#_Toc152328387)

[14.5 Вывод. 86](#_Toc152328388)

[15 Практическая работа №15 87](#_Toc152328389)

[15.1 Цель работы. 87](#_Toc152328390)

[15.2 Задание. 87](#_Toc152328391)

[15.3 Код программы. 87](#_Toc152328392)

[15.4 Тестирование. 92](#_Toc152328393)

[15.5 Вывод. 92](#_Toc152328394)

[16 Практическая работа №16 93](#_Toc152328395)

[16.1 Цель работы. 93](#_Toc152328396)

[16.2 Задание. 93](#_Toc152328397)

[16.3 Код программы. 93](#_Toc152328398)

[16.4 Тестирование. 97](#_Toc152328399)

[16.5 Вывод. 97](#_Toc152328400)

[17 Практическая работа №17 98](#_Toc152328401)

[17.1 Цель работы. 98](#_Toc152328402)

[17.2 Задание. 98](#_Toc152328403)

[17.3 Код программы. 98](#_Toc152328404)

[17.4 Тестирование. 100](#_Toc152328405)

[17.5 Вывод. 100](#_Toc152328406)

[18 Практическая работа №18 101](#_Toc152328407)

[18.1 Цель работы. 101](#_Toc152328408)

[18.2 Задание. 101](#_Toc152328409)

[18.3 Код программы. 101](#_Toc152328410)

[18.4 Тестирование. 105](#_Toc152328411)

[18.5 Вывод. 105](#_Toc152328412)

[19 Практическая работа №19 106](#_Toc152328413)

[19.1 Цель работы. 106](#_Toc152328414)

[19.2 Задание. 106](#_Toc152328415)

[19.3 Код программы. 106](#_Toc152328416)

[19.4 Тестирование. 107](#_Toc152328417)

[19.5 Вывод. 107](#_Toc152328418)

[20 Практическая работа №20 108](#_Toc152328419)

[20.1 Цель работы. 108](#_Toc152328420)

[20.2 Задание. 108](#_Toc152328421)

[20.3 Код программы. 108](#_Toc152328422)

[20.4 Тестирование. 112](#_Toc152328423)

[20.5 Вывод. 112](#_Toc152328424)

[21 Практическая работа №21 113](#_Toc152328425)

[21.1 Цель работы. 113](#_Toc152328426)

[21.2 Задание. 113](#_Toc152328427)

[21.3 Код программы. 113](#_Toc152328428)

[21.4 Тестирование. 123](#_Toc152328429)

[21.5 Вывод. 123](#_Toc152328430)

[22 Практическая работа №22 124](#_Toc152328431)

[22.1 Цель работы. 124](#_Toc152328432)

[22.2 Задание. 124](#_Toc152328433)

[22.3 Код программы. 124](#_Toc152328434)

[22.4 Тестирование. 130](#_Toc152328435)

[22.5 Вывод. 130](#_Toc152328436)

[23 Практическая работа №23 131](#_Toc152328437)

[23.1 Цель работы. 131](#_Toc152328438)

[23.2 Задание. 131](#_Toc152328439)

[23.3 Код программы. 132](#_Toc152328440)

[23.4 Тестирование. 147](#_Toc152328441)

[23.5 Вывод. 147](#_Toc152328442)

[24 Практическая работа №24 148](#_Toc152328443)

[24.1 Цель работы. 148](#_Toc152328444)

[24.2 Задание. 148](#_Toc152328445)

[24.3 Код программы. 149](#_Toc152328446)

[24.4 Тестирование. 179](#_Toc152328447)

[24.5 Вывод. 179](#_Toc152328448)

[25 Список литературы 180](#_Toc152328449)

# Практическая работа №1

## Цель работы.

Знакомство со втроенными функциональными интерфейсами Java. Возможности Java 8. Лямбда-выражения. Области действия, замыквания. Предикаты. Функции. Компараторы.

## Задание.

Имплементировать интерфейс Predicate, определяющий, содержит

ли массив студентов студента с максимальным количеством баллов

(максимальное значение – 100)

Код программы.

package pra1;  
  
public class Student {  
 public Integer id;  
 public Integer ball;  
  
 public Student(Integer id, Integer ball) {  
 this.id = id;  
 this.ball = ball;  
 }  
}

Листинг 1.1 – класс Student.

package pra1;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.function.Predicate;  
  
public class Main {  
  
 public static void main(String[] args) {  
 Predicate<List<Student>> hasMax = (students)->{  
 for(Student s : students){  
 if(s.ball==100) return true;  
 }  
 return false;  
 };  
 var st1 = new Student(1, 50);  
 var st2 = new Student(2, 100);  
 var st3 = new Student(3, 40);  
 var without = new ArrayList<Student>();  
 without.addAll(List.*of*(st1, st3));  
 var with = new ArrayList<Student>();  
 with.addAll(List.*of*(st1,st2,st3));  
 System.out.println(hasMax.test(without));  
 System.out.println(hasMax.test(with));  
 }  
}

Листинг 1.2 – код класса Main.

## Тестирование.



Рисунок 1 - результат работы программы.

## Выводы.

Освоили работу с функциональными интерфейсами в Java.

# Практическая работа №2

## Цель работы.

Работа со Stream API в Java 8.

## Задание.

Сортировка по последней букве фамилии, фильтрация по признаку

«возраст больше, чем вес», сортировка по дате рождения, произведение всех возрастов.

## Код программы.

import java.time.LocalDate;  
  
public class Human {  
 public int age;  
 public String firstName;  
 public String lastName;  
 public LocalDate birthDate;  
 public 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;  
 }  
  
 @Override  
 public String toString() {  
 return "Human{" +  
 "age=" + age +  
 ", firstName='" + firstName + '\'' +  
 ", lastName='" + lastName + '\'' +  
 ", birthDate=" + birthDate +  
 ", weight=" + weight +  
 '}';  
 }  
}

Листинг 2.1 – код класса Human.

import java.time.LocalDate;  
import java.time.Month;  
import java.util.ArrayList;  
import java.util.Comparator;  
import java.util.List;  
import java.util.Optional;  
import java.util.stream.Collectors;  
  
public class HumanWorker {  
 private List<Human> humans;  
 public HumanWorker(){  
 this.humans = new ArrayList<>();  
 var h1 = new Human(20, "Aleksander", "Fridrikh", LocalDate.*of*(2004, Month.AUGUST, 13), 85);  
 var h2 = new Human(40, "Ivan", "Pupkin", LocalDate.*of*(1984, Month.FEBRUARY, 17), 65);  
 var h3 = new Human(13, "Katerina", "Melkaya", LocalDate.*of*(2011, Month.JULY, 3), 48);  
 var h4 = new Human(70, "Vladislav", "Rakov", LocalDate.*of*(1954, Month.MARCH, 27), 60);  
 var h5 = new Human(66, "Vladislava", "Rakova", LocalDate.*of*(1958, Month.JANUARY, 1), 49);  
 var h6 = new Human(34, "Artemon", "Bogomolov", LocalDate.*of*(1990, Month.NOVEMBER, 20), 140);  
 var h7 = new Human(34, "Slava", "Lapov", LocalDate.*of*(1990, Month.MAY, 9), 80);  
 var h8 = new Human(15, "Aleksandr", "Pubertat", LocalDate.*of*(2009, Month.APRIL, 5), 60);  
 var h9 = new Human(18, "Anton", "Checkov", LocalDate.*of*(2006, Month.SEPTEMBER, 14), 75);  
 var h10 = new Human(23, "Maria", "Chernova", LocalDate.*of*(2001, Month.JUNE, 25), 55);  
 this.humans.addAll(List.*of*(h1,h2,h3,h4,h5,h6,h7,h8,h9,h10));  
 }  
 public List<Human> sortByLastSymb(){  
 return this.humans.stream().sorted(Comparator.*comparingInt*(h -> h.lastName.charAt(h.lastName.length() - 1))).collect(Collectors.*toList*());  
 }  
 public List<Human> filterByWeight(){  
 return this.humans.stream().filter(h->h.age>h.weight).collect(Collectors.*toList*());  
 }  
 public List<Human> sortByDate(){  
 return this.humans.stream().sorted(Comparator.*comparing*(h -> h.birthDate)).collect(Collectors.*toList*());  
 }  
 public Integer getMultiplication() throws Exception{  
 return this.humans.stream().map(h->h.age).reduce((h1,h2)->h1\*h2).orElseThrow(()->new Exception("can't get multiplication"));  
 }  
}

Листинг 2.2 – код класса HumanWorker.

public class Main {  
 public static void main(String[] args) {  
 var test = new HumanWorker();  
 System.out.println("SORTED");  
 test.sortByLastSymb().forEach(System.out::println);  
 System.out.println("FILTERING");  
 test.filterByWeight().forEach(System.out::println);  
 System.out.println("SORTED BY DATE");  
 test.sortByDate().forEach(System.out::println);  
 System.out.println("GET MULTIPLICATION");  
 try {  
 System.out.println(test.getMultiplication());  
 }catch (Exception e){  
 System.out.println(e.getMessage());  
 }  
 }  
}

Листинг 2.3 – код класса Main.

## Тестирование.

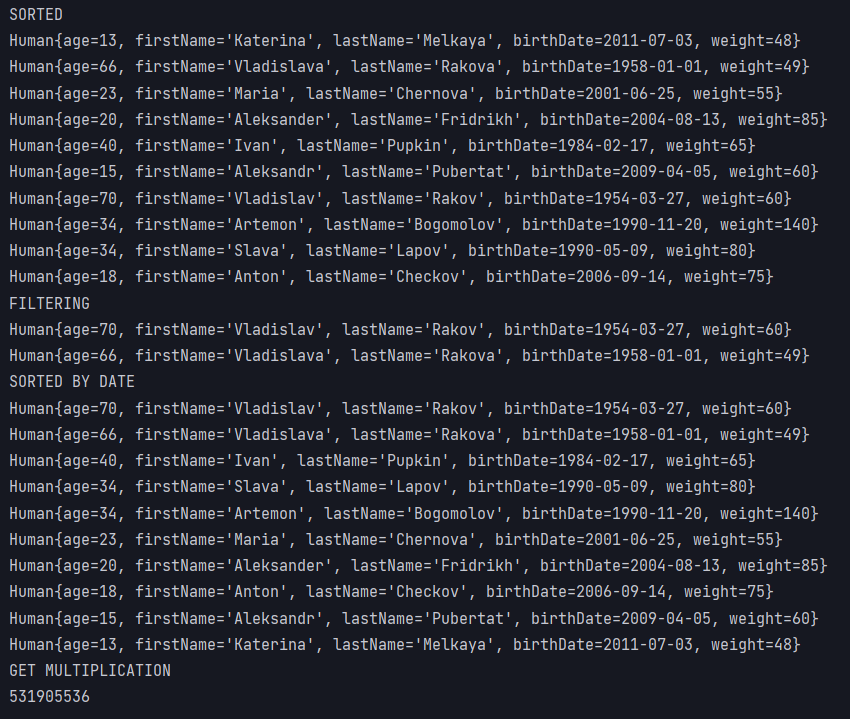


Рисунок 2 - результат работы программы.

## Вывод.

Получил знания и практические навыки по работе с Stream API в Java 8.

# Практическая работа №3

## Цель работы.

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

## Задание.

Реализовать потокобезопасную коллекцию Мap с использованием семафора, а также потокобезопасный Set с использованием ключевого слова synchronized.

## Код программы.

import java.util.HashMap;  
import java.util.Map;  
import java.util.concurrent.Semaphore;  
  
public class MapSemaphore {  
 public static void main(String[] args) {  
 Map<Integer, String> syncMap = new HashMap<>();  
 var semaphore = new Semaphore(1);  
 var first = new Thread(()->{  
 try{  
 semaphore.acquire();  
 System.out.println("First thread");  
 syncMap.put(1, "Clynic");  
 syncMap.put(2, "K");  
 syncMap.put(3, "A");  
 syncMap.put(4, "D");  
 syncMap.keySet().stream().forEach(k->{  
 System.out.println(k+":"+syncMap.get(k));  
 });  
 }catch (Exception e){  
 System.out.println(e.getMessage());  
 }finally{  
 semaphore.release();  
 }  
 });  
 var second = new Thread(()->{  
 try{  
 semaphore.acquire();  
 System.out.println("Second thread");  
 syncMap.put(1, "NIKNOJIsfbh;Op");  
 syncMap.put(2, "A");  
 syncMap.put(3, "L");  
 syncMap.put(4, "O");  
 syncMap.keySet().stream().forEach(k->{  
 System.out.println(k+":"+syncMap.get(k));  
 });  
 }catch (Exception e){  
 System.out.println(e.getMessage());  
 }finally{  
 semaphore.release();  
 }  
 });  
 first.start();  
 second.start();  
 }  
}

Листинг 3.1 – код класса MapSemaphore.

import java.util.HashSet;  
import java.util.Set;  
  
public class SetSync {  
 public static void main(String[] args) {  
 Set<Integer> syncSet = new HashSet<>();  
 var first = new Thread(()->{  
 synchronized (syncSet){  
 System.out.println("First thread");  
 syncSet.add(1);  
 syncSet.add(2);  
 syncSet.add(3);  
 System.out.println(syncSet);  
 }  
 });  
 var second = new Thread(()->{  
 synchronized (syncSet){  
 System.out.println("Second thread");  
 syncSet.add(4);  
 syncSet.add(5);  
 syncSet.add(6);  
 System.out.println(syncSet);  
 }  
 });  
 var third = new Thread(()->{  
 synchronized (syncSet){  
 System.out.println("Third thread");  
 syncSet.add(7);  
 syncSet.add(8);  
 syncSet.add(9);  
 System.out.println(syncSet);  
 }  
 });  
 first.start();  
 second.start();  
 third.start();  
  
 }  
}

Листинг 3.2 – код класса SyncSet.

## Тестирование.

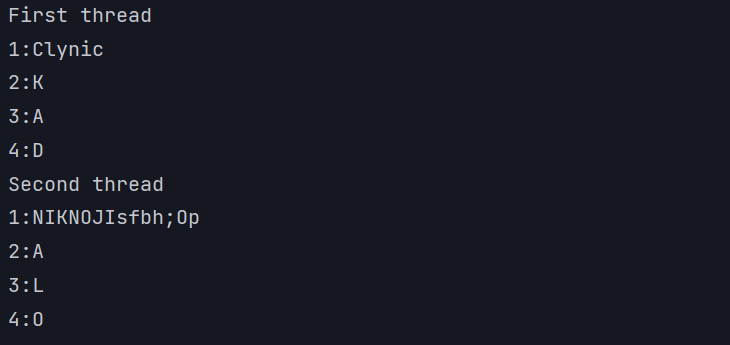


Рисунок 3 - результат работы MapSemaphore.

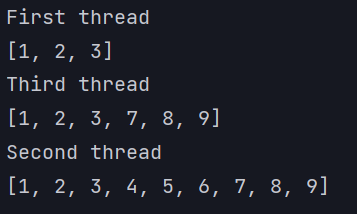


Рисунок 4 - результат работы SyncSet.

## Вывод.

Изучены основы работы с потоками. Реализованы потокобезопасные коллекции.

# Практическая работа №4

## Цель работы.

Работа с ExecutorService, CompletableFuture.

## Задание.

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

## Код программы.

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

Листинг 4.1 – код класса Realization.

public class Main {  
 public static void main(String[] args) {  
 var result = new Realization(5);  
 result.submit(() -> {  
 try {  
 Thread.*sleep*(200);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 System.out.println("We run it");  
 });  
 result.submit(() -> System.out.println("Start"));  
 }  
}

Листинг 4.2 – код класса Main.

## Тестирование.

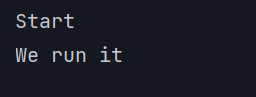


Рисунок 5 – результат тестирования программы.

## Вывод.

Изучена работа с ExecutorService, CompletableFuture.

# Практическая работа №5

## Цель работы.

Познакомиться с паттернами проектирования, их определением и

классификацией. Обзор паттернов GoF. Паттерн Синглтон.

## Задание.

Реализовать паттерн Singleton как минимум 3-мя способами.

## Код программы.

public class Singleton1 {  
 private static Singleton1 INSTANCE = new Singleton1();  
 private Singleton1(){}  
 public void printMessage(){  
 System.out.println("Message from singleton 1");  
 }  
 public static Singleton1 getInstance(){  
 return Singleton1.INSTANCE;  
 }  
}

Листинг 5.1 – код класса Singleton1.

public class Singleton2{  
 private static Singleton2 INSTANCE;  
  
 private Singleton2() {}  
  
 public static Singleton2 getInstance() {  
 if (INSTANCE == null) {  
 INSTANCE = new Singleton2();  
 }  
 return INSTANCE;  
 }  
 public void printMessage(){  
 System.out.println("Message from singleton2");  
 }  
}

Листинг 5.2 – код класса Singleton2.

public class Singleton3 {  
  
 private Singleton3() {  
 }  
 public void printMessage(){  
 System.out.println("Message from singleton 3");  
 }  
 private static class SingletonHolder {  
 public static final Singleton3 HOLDER\_INSTANCE = new Singleton3();  
 }  
  
 public static Singleton3 getInstance() {  
 return SingletonHolder.HOLDER\_INSTANCE;  
 }  
}

Листинг 5.3 – код класса Singleton3.

public class Main {  
 public static void main(String[] args) {  
 var s1 = Singleton1.*getInstance*();  
 s1.printMessage();  
 var s2 = Singleton2.*getInstance*();  
 s2.printMessage();  
 var s3 = Singleton3.*getInstance*();  
 s3.printMessage();  
 }  
}

Листинг 5.4 – код класса Main.

## Тестирование.

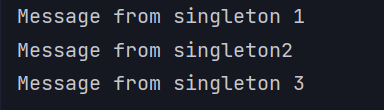


Рисунок 6 – результат тестирования программы.

## Вывод.

Освоен паттерн Singleton.

# Практическая работа №6

## Цель работы.

Знакомство с реализацией порождающих паттернов проектирования.

## Задание.

Написать реализацию паттернов «Фабричных методов», «Абстрактная фабрика», «Строитель», «Прототип».

## Код программы.

public interface Prototype {  
 Prototype clone();  
}

Листинг 6.1 – код интерфейса «Prototype».

package prototype;  
  
public class Dog implements Prototype{  
 private String name;  
 private int age;  
 public Dog(String name, int age){  
 this.name = name;  
 this.age = age;  
 }  
 public void printInfo(){  
 System.out.println("Dog's name is "+this.name+"\n" +  
 "Dog's age is "+this.age);  
 }  
  
 @Override  
 public Dog clone() {  
 return new Dog(this.name, this.age);  
 }  
}

Листинг 6.2 – код класса Dog.

import abstractFabric.CarFactory;  
import abstractFabric.LadaFactory;  
import abstractFabric.MercedesFactory;  
import builder.ConcreteBuilder;  
import builder.Director;  
import fabricMethod.ConcreteCreator;  
import fabricMethod.Creator;  
import fabricMethod.Product;  
import prototype.Dog;  
  
public class Main {  
 public static void main(String[] args) {  
 //fabric method  
 Creator creator = new ConcreteCreator();  
 Product product = creator.create();  
 product.perform();  
 //abstract factory  
 CarFactory fabric = new MercedesFactory();  
 fabric.createCoupe().drive();  
 fabric.createSedan().drive();  
 fabric = new LadaFactory();  
 fabric.createCoupe().drive();  
 fabric.createSedan().drive();  
 //builder  
 var point = new Director(new ConcreteBuilder()).createPoint();  
 System.out.println(point);  
 //prototype  
 var dog1 = new Dog("Alice", 12);  
 dog1.printInfo();  
 var clone = dog1.clone();  
 clone.printInfo();  
 }  
}

Листинг 6.3 – код класса Main.

## Тестирование.

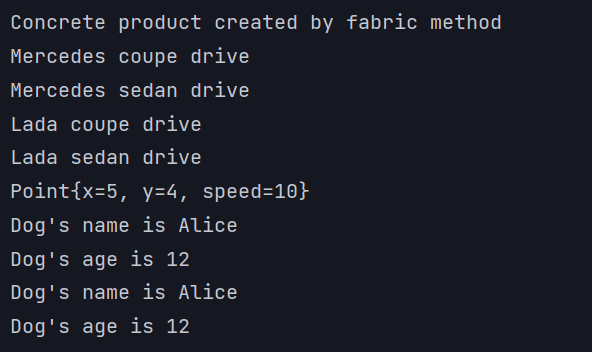


Рисунок 6 – результат тестирования программы.

## Вывод.

Реализованы основные порождающие паттерны проектирования.

# Практическая работа №7

## Цель.

Реализация структурных паттернов проектирования.

## Задание.

Реализовать паттерны «Фасад» и «Заместитель».

## Код программы.

package facade;  
  
public class Database {  
 public void startDatabase(){  
 System.out.println("Starting database");  
 }  
 public void shutdownDatabase(){  
 System.out.println("Shutdown database");  
 }  
}

Листинг 7.1 – код класса Database.

package facade;  
  
public class FrontendOnReact {  
 public void startFrontend(){  
 System.out.println("Started frontend server");  
 }  
 public void shutdownFrontend(){  
 System.out.println("Shutdown frontend");  
 }  
}

Листинг 7.2 – код класса FrontendOnReact.

package facade;  
  
public class HttpServer {  
 public void startServer(){  
 System.out.println("Starting http server");  
 }  
 public void shutdownServer(){  
 System.out.println("Shutdown http server");  
 }  
}

Листинг 7.3 – код класса HttpServer.

package facade;  
  
public class SystemFacade {  
 private Database db;  
 private HttpServer server;  
 private FrontendOnReact frontendOnReact;  
  
 public SystemFacade(Database db, HttpServer server, FrontendOnReact frontendOnReact) {  
 this.db = db;  
 this.server = server;  
 this.frontendOnReact = frontendOnReact;  
 }  
 public void startSystem(){  
 db.startDatabase();  
 server.startServer();  
 frontendOnReact.startFrontend();  
 }  
 public void shutdownSystem(){  
 db.shutdownDatabase();  
 server.shutdownServer();  
 frontendOnReact.shutdownFrontend();  
 }  
}

Листинг 7.4 – код класса SystemFacade.

package proxy;  
  
public interface Website {  
 String getAnswer(String request);  
}

Листинг 7.5 – код интерфейса Website.

package proxy;  
  
public class RealWebsite implements Website{  
 @Override  
 public String getAnswer(String request) {  
 try{  
 Thread.*sleep*(3000);  
 }catch (Exception e){  
 System.out.println(e.getMessage());  
 }  
 return "Hello "+request;  
 }  
}

Листинг 7.6 – код класса RealWebsite.

import java.util.HashMap;  
import java.util.Map;  
  
public class CachingProxy implements Website{  
 private final RealWebsite website = new RealWebsite();  
 private Map<String, String> cache = new HashMap<>();  
  
 @Override  
 public String getAnswer(String request) {  
 if(cache.get(request)==null){  
 var result = website.getAnswer(request);  
 cache.put(request, result);  
 return result;  
 }else{  
 return cache.get(request);  
 }  
 }  
}

Листинг 7.7 – код класса CachingProxy.

import facade.Database;  
import facade.FrontendOnReact;  
import facade.HttpServer;  
import facade.SystemFacade;  
import proxy.CachingProxy;  
  
public class Main {  
 public static void main(String[] args) throws InterruptedException {  
 SystemFacade system = new SystemFacade(  
 new Database(),  
 new HttpServer(),  
 new FrontendOnReact()  
 );  
 system.startSystem();  
 Thread.*sleep*(3000);  
 system.shutdownSystem();  
 //proxy  
 var proxy = new CachingProxy();  
 System.out.println(proxy.getAnswer("Vanya"));  
 System.out.println(proxy.getAnswer("Vanya"));  
 }  
}

Листинг 7.8 – код класса Main.

## Тестирование.

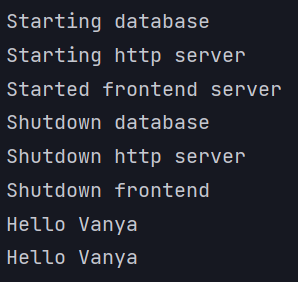


Рисунок 7 – результат тестирование программы.

## Вывод.

Изучены основные структурные паттерны проектирования.

# Практическая работа №8

## Цель работы.

Реализация поведенческих паттернов проектирования.

## Задание.

Реализовать паттерны «Посредник» и «Итератор».

## Код программы.

package iterator;  
  
public interface Aggregate {  
 Iterator createIterator();  
}

Листинг 8.1 – код интерфейса Aggregate.

package iterator;  
  
public interface Iterator {  
 void next();  
}

Листинг 8.2 – код интерфейса Iterator.

package iterator;  
  
public class ConcreteIterator implements Iterator{  
 private int[] array;  
 private int current = 0;  
 public ConcreteIterator(int[] array){  
 this.array = array;  
 }  
 @Override  
 public void next() {  
 if(current<this.array.length){  
 System.out.println(this.array[current]);  
 current++;  
 }else{  
 System.out.println("Iterator finished");  
 }  
 }  
}

Листинг 8.3 – код класса ConcreteIterator.

package iterator;  
  
public class ConcreteAggregate implements Aggregate{  
 private int[] array = {1,2,3,4,5,6};  
 private Iterator iterator = new ConcreteIterator(this.array);  
 @Override  
 public Iterator createIterator() {  
 return iterator;  
 }  
}

Листинг 8.4 – код класса ConcreteAggregate.

package mediator;  
  
public interface Mediator {  
 void addUser(User u);  
 void sendMessage(String msg);  
}

Листинг 8.5 – код интерфейса Mediator.

package mediator;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class MessageBroker implements Mediator{  
 private List<User> users;  
 public MessageBroker(){  
 this.users = new ArrayList<>();  
 }  
 @Override  
 public void addUser(User u) {  
 this.users.add(u);  
 }  
  
 @Override  
 public void sendMessage(String msg) {  
 for(User u: this.users){  
 System.out.println("User "+u.name + " has received message: "+msg);  
 }  
 }  
}

Листинг 8.6 – код класса MessageBroker.

package mediator;  
  
public class Student extends User{  
 public Student(Mediator mediator, String name){  
 super(mediator, name);  
 }  
 @Override  
 public void send(String msg) {  
 this.mediator.sendMessage(msg+" from Student");  
 }  
}

Листинг 8.7 – код класса Student.

package mediator;  
  
public class Teacher extends User {  
 public Teacher(Mediator mediator, String name){  
 super(mediator, name);  
 }  
 @Override  
 public void send(String msg) {  
 this.mediator.sendMessage(msg+" from Teacher");  
 }  
}

Листинг 8.8 – код класса Teacher.

package mediator;  
  
abstract public class User {  
 protected Mediator mediator;  
 public String name;  
 public User(Mediator mediator, String name){  
 this.mediator = mediator;  
 this.name = name;  
 }  
 public abstract void send(String msg);  
}

Листинг 8.9 – код класса User.

import iterator.ConcreteAggregate;  
import mediator.MessageBroker;  
import mediator.Student;  
import mediator.Teacher;  
import mediator.User;  
  
public class Main {  
 public static void main(String[] args) {  
 var aggregate = new ConcreteAggregate();  
 var iterator = aggregate.createIterator();  
 iterator.next();  
 iterator.next();  
 iterator.next();  
 iterator.next();  
 iterator.next();  
 //mediator  
 MessageBroker msgBroker = new MessageBroker();  
 Teacher u1 = new Teacher(msgBroker, "Alex");  
 Student u2 = new Student(msgBroker, "Ivan");  
 Student u3 = new Student(msgBroker, "Nikita");  
 Teacher u4 = new Teacher(msgBroker, "Alim");  
 msgBroker.addUser(u1);  
 msgBroker.addUser(u2);  
 msgBroker.addUser(u3);  
 msgBroker.addUser(u4);  
 u1.send("Message from User with name:"+u1.name);  
 }  
}

Листинг 8.10 – код класса Main.

## Тестирование.

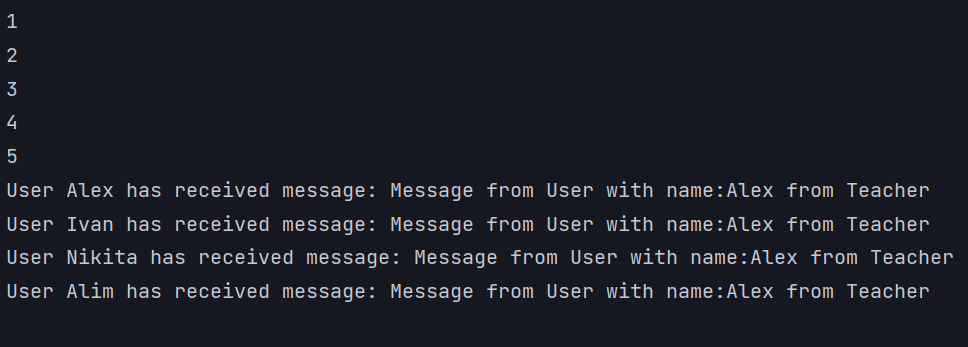


Рисунок 8 – результат тестирования программы.

## Вывод.

Изучены основные поведенческие паттерны проектирования.

# Практические работы №9

## Цель работы.

Знакомство с системой сборки Gradle.

## Задание.

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

plugins **{** id 'java'  
**}**group = 'org.example'  
  
repositories **{** mavenCentral()  
**}**dependencies **{** testImplementation platform('org.junit:junit-bom:5.9.1')  
 testImplementation 'org.junit.jupiter:junit-jupiter'  
**}**test **{** useJUnitPlatform()  
**}**task createJar(type: Jar) **{** archiveFileName = 'practic9.jar'  
 from sourceSets.main.output  
**}**task moveJar(type: Copy) **{** dependsOn createJar  
 from 'build/libs'  
 into 'dock'  
**}**task buildDockerImage(type: Exec) **{** dependsOn moveJar  
 workingDir 'dock'  
 commandLine 'docker', 'build', '-t', 'new-image', '.'  
**}**task dockerRun(type: Exec) **{** dependsOn buildDockerImage  
 commandLine 'docker', 'run', '-d', '-p', '8080:8080', 'new-image'  
**}**

Листинг 9.1 – код build.gradle.

## Тестирование.



Рисунок 9 – результат тестирования программы.

## Вывод.

Изучена система сборки Gradle.

# Практическая работа №10

## Цель работы.

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

## Задание.

Создать приложение, в котором создается ApplicationContext и из него берётся бин с названием, переданным в качестве аргумента к приложению, и вызывается метод интерфейса, который он имплементирует. Нужно создать по одному бину для каждого класса, определить им название. Проверить, что вызывается при вводе названия каждого из бинов. Классы и интерфейс определяются в соответствии с вариантом индивидуального задания: Интерфейс Programmer с методом doCoding(), его имплементации: Junior, Middle, Senior

## Код программы.

package ru.example;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class BeanConfig {  
 @Bean  
 public Junior junProgrammer () {  
 return new Junior();  
 }  
  
 @Bean  
 public Middle midProgrammer () {  
 return new Middle();  
 }  
  
 @Bean  
 public Senior senProgrammer () {  
 return new Senior();  
 }  
  
 @Bean("jun")  
 public Programming startProgrammingJun() {  
 Programming programming = new Programming();  
 programming.setType(junProgrammer());  
 return programming;  
 }  
  
 @Bean("mid")  
 public Programming startProgrammingMid() {  
 Programming programming = new Programming();  
 programming.setType(midProgrammer());  
 return programming;  
 }  
  
 @Bean("sen")  
 public Programming startProgrammingSen() {  
 Programming programming = new Programming();  
 programming.setType(senProgrammer());  
 return programming;  
 }  
}

Листинг 10.1 – код класса BeanConfig.

## Тестирование.

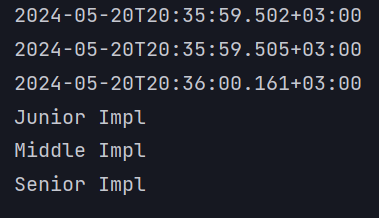


Рисунок 10 – результат тестирование программы.

## Вывод.

Изучены основные концепты работы с фреймворком Spring.

# Практическое задание №11

## Цель работы.

Разобраться с использованием Spring boot.

## Задание.

Создать приложение с использованием Spring Boot Starter Initializr (https://start.spring.io/) с такими зависимостями:

– Spring Web;

– Lombok;

– Validation;

– Spring boot Actuator.

Запустить приложение и удостовериться, что не появилось никаких ошибок. Добавить все эндпоинты в Actuator, сделать HTTP-запрос на проверку состояния приложения. Собрать jar-файл приложения, запустить и проверить состояние при помощи REST-запроса

## Код программы.

package ru.example;  
  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
@RequestMapping  
public class RestHello {  
  
 @GetMapping  
 public String hello() {  
 return String.*format*("Hello world");  
 }  
}

Листинг 11.1 – код класса RestHello.

## Тестиривание.

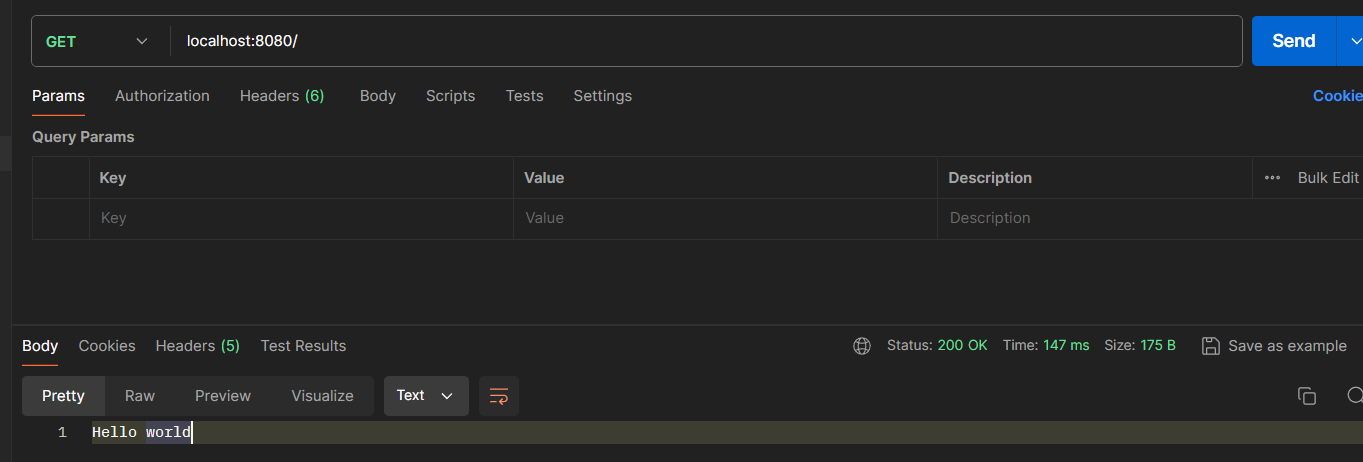


Рисунок 11.1 – результат тестирования программы.

## Вывод.

Разобрались с использованием Spring boot.

# Практическая работа №12

## Цель работы.

Работа с жизненным циклом компонентов. Аннотации PostConstruct, PreDestroy.

## Задание.

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

## Код программы.

package ru.example;  
  
import jakarta.annotation.PostConstruct;  
import jakarta.annotation.PreDestroy;  
import lombok.Getter;  
import lombok.Setter;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.stereotype.Component;  
  
import java.io.\*;  
  
  
@Getter  
@Setter  
@Slf4j  
@Component  
public class ComponentClass {  
 private static final String from = "C:\\study\\jaba\\practic12\\first-file.txt";  
 private static final String to = "C:\\study\\jaba\\practic12\\second-file.txt";  
  
 @PostConstruct  
 public void postConstruct() {  
 log.info("Started!");  
 }  
  
 @PreDestroy  
 public void preDestroy() throws IOException {  
 BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(to));  
 if (new File(from).exists()) {  
 BufferedReader bufferedReader = new BufferedReader(new FileReader(from));  
 StringBuilder text = new StringBuilder();  
 String line;  
 while ((line = bufferedReader.readLine()) != null) {  
 text.append(line);  
 }  
 bufferedWriter.write(String.*valueOf*(text.hashCode()));  
 bufferedReader.close();  
 } else {  
 bufferedWriter.write("null");  
 }  
 bufferedWriter.close();  
 }  
}

Листинг 12.1 – код класса ComponentClass.

## Тестирование.

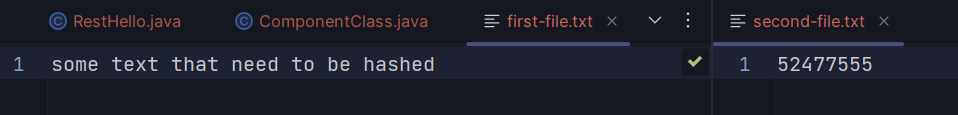


Рисунок 12 – результат тестирования программы.

## Вывод.

Разобрался в работе с жизненным циклом компонентов, аннотациями PreDestroy, PostConstruct.

# Практическая работа №13

## Цель работы.

Конфигурирование приложения. Environment.

## Задание.

Создать файл application.yml в папке resources, добавить в него такие свойства:

– student.name – имя студента;

– student.last\_name – фамилия студента;

– student.group – название группы студента.

При запуске приложения выведите данные свойства в консоль при помощи интерфейса Environment или аннотации Value.

## Код программы.

package ru.example;  
  
import jakarta.annotation.PostConstruct;  
import lombok.RequiredArgsConstructor;  
import lombok.ToString;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
@Slf4j  
@Component  
@RequiredArgsConstructor  
@ToString  
public class User {  
  
 @Value("${student.name}")  
 private final String name;  
  
 @Value("${student.last\_name}")  
 private final String last\_name;  
  
 @Value("${student.group}")  
 private final String group;  
  
 @PostConstruct  
 public void start() {  
 log.info("create user: {}", this);  
 }  
}

Листинг 13.1 – код класса User.

student:  
 name: Aleksander  
 last\_name: Fridrikh  
 group: IKBO-20-22

Листинг 13.2 – код файла application.yml.

## Тестирование.

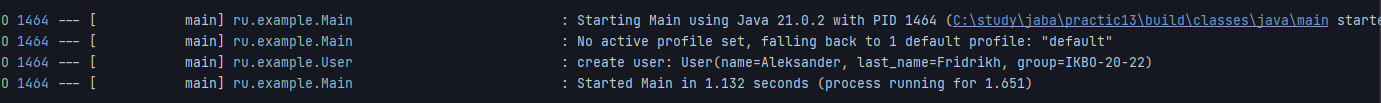


Рисунок 13.1 – результат тестирования программы.

## Вывод.

Разобрался с конфигурированием приложения, Environment.

# Практическая работа №14

## Цель работы.

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

## Задание.

Создать отдельный репозиторий Git. Создать простой html-документ, который будет содержать вашу фамилию, имя, номер группы, номер варианта. Создать контроллер, который будет возвращать данный статический документ при переходе на url «/home». Выполнить задание в зависимости с вариантом индивидуального задания: Создать класс Card с полями cardNumber, code. Создать класс Bank с полями name, address. Создать классы-контроллеры для создания, удаления объектов и получения всех объектов каждого типа. Сами объекты хранить в памяти.

## Код программы.

package ru.practic14;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.GetMapping;  
  
@Controller  
public class HomeController {  
 @GetMapping("/home")  
 public String home(){  
 return "index";  
 }  
}

Листинг 14.1 – код класса HomeController

package ru.practic14;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.DeleteMapping;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.ResponseBody;  
  
import java.util.ArrayList;  
import java.util.List;  
  
@Controller  
@RequestMapping("/cards")  
public class CardController {  
 private List<Card> cards = new ArrayList<>();  
  
 @PostMapping("/add")  
 @ResponseBody  
 public Card createCard(@RequestBody Card card) {  
 cards.add(card);  
 return card;  
 }  
  
 @GetMapping("/")  
 @ResponseBody  
 public List<Card> getAllCards() {  
 return cards;  
 }  
  
 @DeleteMapping("/{index}")  
 @ResponseBody  
 public Card deleteCard(@PathVariable int index) {  
 if (index >= 0 && index < cards.size()) {  
 return cards.remove(index);  
 } else {  
 return null;  
 }  
 }  
}

Листинг 14.2 – код класса CardController.

package ru.practic14;  
  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.ArrayList;  
import java.util.List;  
  
@RestController  
@RequestMapping("/banks")  
public class BankController {  
 private List<Bank> bankList = new ArrayList<>();  
  
 @PostMapping("/")  
 public Bank createBank(@RequestBody Bank bank) {  
 bankList.add(bank);  
 return bank;  
 }  
  
 @GetMapping("/")  
 public List<Bank> getAllBanks() {  
 return bankList;  
 }  
  
 @DeleteMapping("/{index}")  
 public Bank deleteBank(@PathVariable int index) {  
 if (index >= 0 && index < bankList.size()) {  
 return bankList.remove(index);  
 } else {  
 return null;  
 }  
 }  
}

Листинг 14.3 – код класса BankController.

package ru.practic14;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.Setter;  
  
@Getter  
@Setter  
@AllArgsConstructor  
public class Card {  
 private Integer cardNumber;  
 private Integer code;  
}

Листинг 14.4 – код класса Card.

package ru.practic14;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.Setter;  
  
@Getter  
@Setter  
@AllArgsConstructor  
public class Bank {  
 private String name;  
 private String address;  
}

Листинг 14.5 – код класса Bank.

## Тестирование.

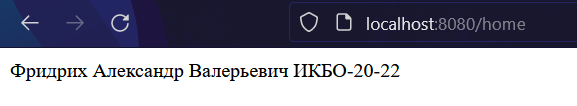


Рисунок 14.1 – результат тестирования программы.

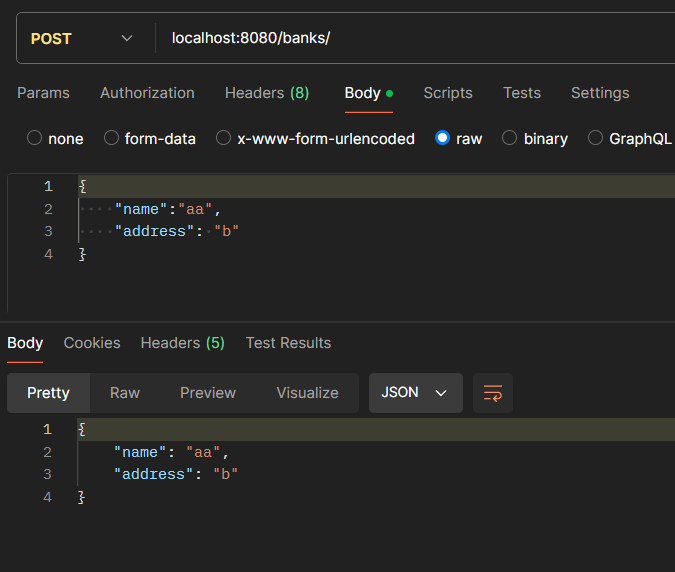


Рисунок 14.2 – результат тестирования программы.

## Вывод.

Познакомился с Spring MVC, поработал с Rest API в Spring.

# Практическая работа №15

## Цель работы.

Использование Hibernate в Spring framework.

## Задание.

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

## Код программы.

package ru.practic15;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Entity  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
public class Card {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 private Integer cardNumber;  
 private Integer code;  
}

Листинг 15.1 – код класса Card.

package ru.practic15;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
import ru.practic15.Card;  
  
  
@Repository  
public interface CardRepository extends JpaRepository<Card, Long> {  
}

Листинг 15.2 – код класса CardRepository.

package ru.practic15;  
  
import lombok.RequiredArgsConstructor;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.DeleteMapping;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.ResponseBody;  
  
import java.util.ArrayList;  
import java.util.List;  
  
@Controller  
@RequestMapping("/cards")  
@RequiredArgsConstructor  
public class CardController {  
 private final CardRepository cardRepository;  
  
 @PostMapping("/")  
 @ResponseBody  
 public Card createCard(@RequestBody Card card) {  
 cardRepository.save(card);  
 return card;  
 }  
  
 @GetMapping("/")  
 @ResponseBody  
 public List<Card> getAllCards() {  
 return cardRepository.findAll();  
 }  
  
 @DeleteMapping("/{id}")  
 @ResponseBody  
 public void deleteCard(@PathVariable long id) {  
 cardRepository.deleteById(id);  
 }  
}

Листинг 15.4 – код класса CardController.

## Тестирование.

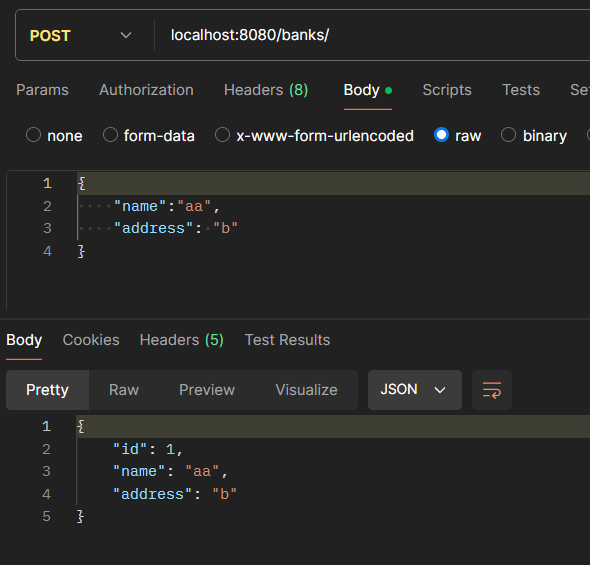


Рисунок 15 – результат тестирования программы.

## Вывод.

Объекты сохраняются в базе данных PostgreSQL.

# Практическая работа №16

## Цель работы.

Изучение видов связей между сущностями в Hibernate. Использование транзакций.

## Задание.

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

## Код программы.

package ru.practic16;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
import jakarta.persistence.JoinColumn;  
import jakarta.persistence.ManyToOne;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Entity  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
public class Card {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 private Integer cardNumber;  
 private Integer code;  
 @ManyToOne  
 @JoinColumn(name="bank\_id", nullable = false)  
 private Bank bank;  
  
 @Override  
 public String toString() {  
 return "Card{" +  
 "id=" + id +  
 ", cardNumber=" + cardNumber +  
 ", code=" + code +  
 ", bank=" + bank +  
 '}';  
 }  
}

Листинг 16.1 – код класса Card.

package ru.practic15;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
package ru.practic16;  
  
import jakarta.persistence.Column;  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
import jakarta.persistence.OneToMany;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.util.List;  
  
@Entity  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
public class Bank {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 @Column(name = "bank\_id")  
 private Long id;  
 private String name;  
 private String address;  
 @OneToMany(mappedBy = "bank")  
 private List<Card> cards;  
  
 @Override  
 public String toString() {  
 return "Bank{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", address='" + address + '\'' +  
 '}';  
 }  
}

Листинг 16.2 – код класса Bank.

package ru.practic16;  
  
import jakarta.transaction.Transactional;  
import lombok.RequiredArgsConstructor;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.DeleteMapping;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.ResponseBody;  
  
import java.util.List;  
import java.util.stream.Collectors;  
  
@Controller  
@RequestMapping("/cards")  
@RequiredArgsConstructor  
@Transactional  
public class CardController {  
 private final CardRepository cardRepository;  
 private final BankRepository bankRepository;  
 @PostMapping("/{bankId}")  
 @ResponseBody  
 public String createCard(@RequestBody Card card, @PathVariable("bankId") Long bankId) {  
 var bank = bankRepository.findById(bankId).orElseThrow();  
 card.setBank(bank);  
 cardRepository.save(card);  
 return card.toString();  
 }  
  
 @GetMapping("/")  
 @ResponseBody  
 public List<String> getAllCards() {  
 return cardRepository.findAll().stream().map(Card::toString).collect(Collectors.*toList*());  
 }  
  
 @DeleteMapping("/{id}")  
 @ResponseBody  
 public void deleteCard(@PathVariable long id) {  
 cardRepository.deleteById(id);  
 }  
}

Листинг 16.3 – код класса CardController.

## Тестирование.

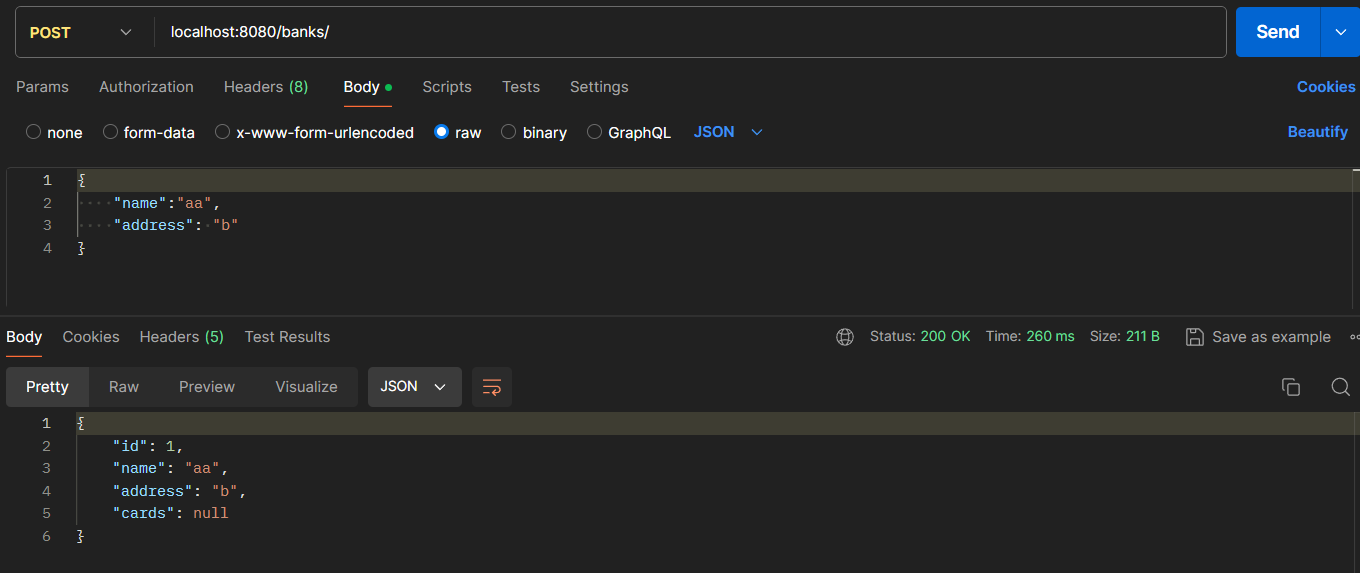


Рисунок 16.1 – результат работы программы.

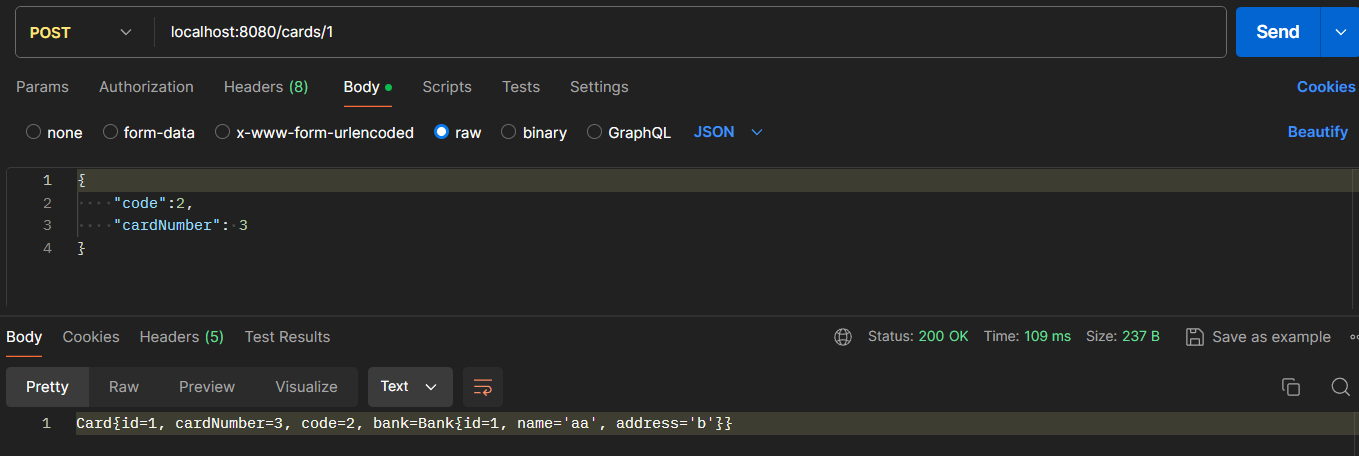


Рисунок 16.2 – результат работы программы.

## Вывод.

Изучил связи между сущностями в Hibernate, использовал транзакции.

# Практическая работа №17

## Цель работы.

Знакомство с Criteria API в Hibernate.

## Задание.

Добавить возможность фильтрации по всем полям всех классов сиспользованием Criteria API в Hibernate для программы из предыдущего задания. Добавить эндпоинты для каждой фильтрации.

## Код программы.

package ru.practic17;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
  
import java.util.Optional;  
  
public interface BankRepository extends JpaRepository<Bank, Long> {  
 @Query("select b from Bank b where b.name=?1")  
 Optional<Bank> getBankByName(String name);  
}

Листинг 17.1 – код класса BankRepository.

package ru.practic17;  
  
import lombok.RequiredArgsConstructor;  
import org.springframework.web.bind.annotation.DeleteMapping;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/banks")  
@RequiredArgsConstructor  
public class BankController {  
 private final BankRepository bankRepository;  
  
 @PostMapping("/")  
 public Bank createBank(@RequestBody Bank bank) {  
 return bankRepository.save(bank);  
 }  
  
 @GetMapping("/{name}")  
 public Bank getByName(@PathVariable("name") String name){  
 return bankRepository.getBankByName(name).orElseThrow();  
 }  
 @GetMapping("/")  
 public List<Bank> getAllBanks() {  
 return bankRepository.findAll();  
 }  
  
 @DeleteMapping("/{id}")  
 public void deleteBank(@PathVariable long id) {  
 bankRepository.deleteById(id);  
 }  
  
}

Листинг 17.2 – код класса BankController.

## Тестирование.

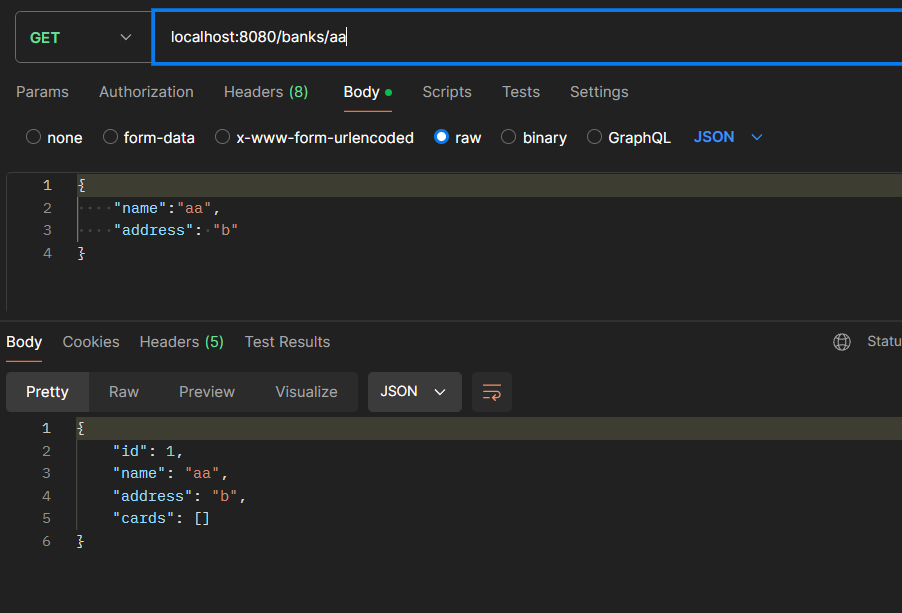


Рисунок 17 – результат работы программы.

## Вывод.

Познакомился с Criteria API в Hibernate.

# Практическая работа №18

## Цель работы.

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

## Задание.

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

## Код программы.

package ru.practic19;  
  
import lombok.RequiredArgsConstructor;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
@RequiredArgsConstructor  
public class BankService {  
 private final BankRepository bankRepository;  
 public List<Bank> getBanks(){  
 return bankRepository.findAll();  
 }  
 public Bank findByName(String name){  
 return bankRepository.getBankByName(name).orElseThrow();  
 }  
 public Bank findById(Long id){  
 return bankRepository.findById(id).orElseThrow();  
 }  
 public Bank saveBank(Bank bank){  
 return bankRepository.save(bank);  
 }  
 public void deleteById(Long id){  
 bankRepository.deleteById(id);  
 }  
}

Листинг 18.1 – код класса BankService.

package ru.practic19;  
  
  
import lombok.RequiredArgsConstructor;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
@RequiredArgsConstructor  
public class CardService {  
 private final CardRepository cardRepository;  
 private final BankRepository bankRepository;  
 public String saveCard(Card card, Long bankId){  
 var bank = bankRepository.findById(bankId).orElseThrow();  
 card.setBank(bank);  
 cardRepository.save(card);  
 return card.toString();  
 }  
  
 public List<Card> getCards(){  
 return cardRepository.findAll();  
 }  
 public void deleteById(Long cardId){  
 cardRepository.deleteById(cardId);  
 }  
}

Листинг 18.2 – код класса CardService.

package ru.practic19;  
  
import lombok.RequiredArgsConstructor;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.DeleteMapping;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.ResponseBody;  
  
import java.util.List;  
import java.util.stream.Collectors;  
  
@Controller  
@RequestMapping("/cards")  
@RequiredArgsConstructor  
public class CardController {  
 private final CardService cardService;  
 private final CardRepository cardRepository;  
 private final BankRepository bankRepository;  
  
 @PostMapping("/{bankId}")  
 @ResponseBody  
 public String createCard(@RequestBody Card card, @PathVariable("bankId") Long bankId) {  
 return cardService.saveCard(card, bankId);  
 }  
  
 @GetMapping("/")  
 @ResponseBody  
 public List<String> getAllCards() {  
 return cardService.getCards().stream().map(Card::toString).collect(Collectors.*toList*());  
 }  
  
 @DeleteMapping("/{id}")  
 @ResponseBody  
 public void deleteCard(@PathVariable long id) {  
 cardService.deleteById(id);  
 }  
}

Листинг 18.3 – код класса CardController.

package ru.practic19;  
  
import lombok.RequiredArgsConstructor;  
import org.springframework.web.bind.annotation.DeleteMapping;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/banks")  
@RequiredArgsConstructor  
public class BankController {  
 private final BankService bankService;  
 @PostMapping("/")  
 public Bank createBank(@RequestBody Bank bank) {  
 return bankService.saveBank(bank);  
 }  
  
 @GetMapping("/{name}")  
 public Bank getByName(@PathVariable("name") String name){  
 return bankService.findByName(name);  
 }  
 @GetMapping("/")  
 public List<Bank> getAllBanks() {  
 return bankService.getBanks();  
 }  
  
 @DeleteMapping("/{id}")  
 public void deleteBank(@PathVariable long id) {  
 bankService.deleteById(id);  
 }  
  
}

Листинг 18.4 – код класса BankController.

## Тестирование.

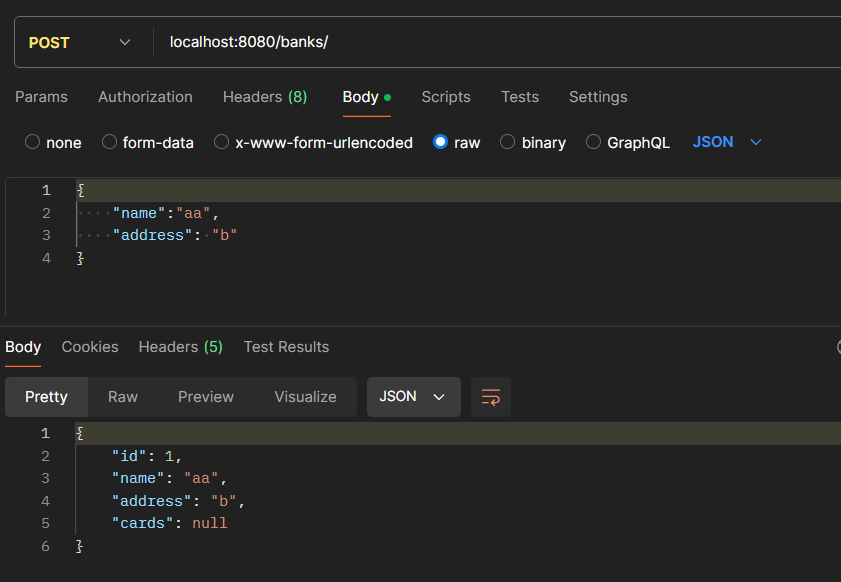


Рисунок 18 – результат работы программы.

## Вывод.

Познакомился с репозиториями и сервисами, реализовал в проекте. Повзаимодействовал с Spring Data JPA.

# Практическая работа №19

## Цель работы.

Знакомство с логированием с использованием Logback в Spring.

## Задание.

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

## Код программы.

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

Листинг 19.1 – код файла logback.xml.

package ru.practic19;  
  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
@RequiredArgsConstructor  
@Slf4j  
public class BankService {  
 private final BankRepository bankRepository;  
 public List<Bank> getBanks(){  
 log.info("finding all banks");  
 return bankRepository.findAll();  
 }  
  
 public Bank findByName(String name){  
 log.info("finding bank by name");  
 return bankRepository.getBankByName(name).orElseThrow();  
 }  
 public Bank findById(Long id){  
 log.info("finding bank by id");  
 return bankRepository.findById(id).orElseThrow();  
 }  
 public Bank saveBank(Bank bank){  
 log.info("saving bank");  
 return bankRepository.save(bank);  
 }  
  
 public void deleteById(Long id){  
 log.info("deleting bank");  
 bankRepository.deleteById(id);  
 }  
}

Листинг 19.2 – код класса BankService.

## Тестирование.

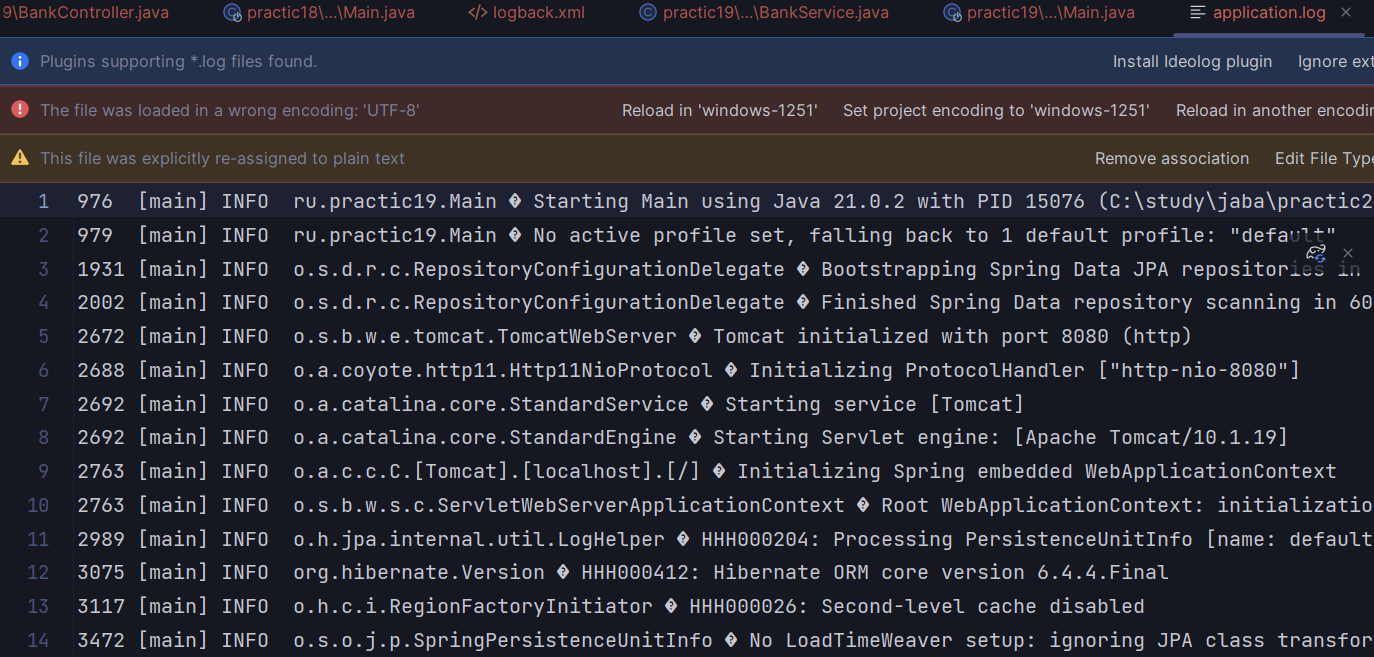


Рисунок 19 – результат работы программы.

## Вывод.

Познакомился с логированием Logback в Spring.

# Практическая работа №20

## Цель работы.

Использование Spring AOP. Pointcut, JoinPoint, Advice.

## Задание.

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

## Код программы.

package ru.practic20;  
  
import lombok.extern.slf4j.Slf4j;  
import org.aspectj.lang.JoinPoint;  
import org.aspectj.lang.annotation.After;  
import org.aspectj.lang.annotation.Aspect;  
import org.aspectj.lang.annotation.Before;  
import org.springframework.stereotype.Component;  
  
@Aspect  
@Component  
@Slf4j  
public class MethodExecutionTimeAspect {  
 private long startTime;  
  
 @Before("execution(\* ru.practic20.BankService.\*(..)) || execution(\* ru.practic20.CardService.\*(..))")  
 public void beforeMethodExecution() {  
 startTime = System.*currentTimeMillis*();  
 }  
  
 @After("execution(\* ru.practic20.BankService.\*(..)) || execution(\* ru.practic20.CardService.\*(..))")  
 public void afterMethodExecution(JoinPoint joinPoint) {  
 long executionTime = System.*currentTimeMillis*() - startTime;  
 log.info("Method" + joinPoint.getSignature() + " was executed in " + executionTime + " ms");  
 }  
}

Листинг 20.1 – код класса MethodExecutionTimeAspect.

## Тестирование.

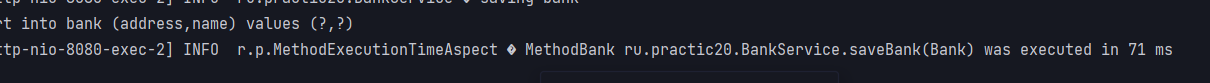


Рисунок 20 – результат работы программы.

## Вывод.

Использованил Spring AOP. Pointcut, JoinPoint, Advice.

# Практическая работа №21

## Цель работы.

Проксирование. Аннотация Transactional. Аннотация Async.

## Задание.

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

## Код программы.

package ru.practic22;  
  
import lombok.AllArgsConstructor;  
import org.springframework.mail.SimpleMailMessage;  
import org.springframework.scheduling.annotation.Async;  
import org.springframework.stereotype.Service;  
import org.springframework.mail.javamail.JavaMailSender;  
@Service  
@AllArgsConstructor  
public class EmailService {  
 private final JavaMailSender javaMailSender;  
 @Async  
 public void saveBank(Bank bank) {  
 SimpleMailMessage mailMessage = new SimpleMailMessage();  
 mailMessage.setFrom("shiningsuffer@gmail.com");  
 mailMessage.setTo("fridrix.aleksandr.2004@mail.ru");  
 mailMessage.setSubject("Message from spring");  
 mailMessage.setText(bank.toString());  
 javaMailSender.send(mailMessage);  
 }  
 @Async  
 public void saveCard(Card card) {  
 SimpleMailMessage mailMessage = new SimpleMailMessage();  
 mailMessage.setFrom("shiningsuffer@gmail.com");  
 mailMessage.setTo("fridrix.aleksandr.2004@mail.ru");  
 mailMessage.setSubject("Message from spring");  
 mailMessage.setText(card.toString());  
 javaMailSender.send(mailMessage);  
 }  
}

Листинг 21.1 – код интерфейса EmailService.

package ru.practic22;  
  
  
import jakarta.transaction.Transactional;  
import lombok.RequiredArgsConstructor;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
@RequiredArgsConstructor  
@Transactional  
public class CardService {  
 private final CardRepository cardRepository;  
 private final BankRepository bankRepository;  
 private final EmailService emailService;  
  
 public String saveCard(Card card, Long bankId) {  
 emailService.saveCard(card);  
 var bank = bankRepository.findById(bankId).orElseThrow();  
 card.setBank(bank);  
 cardRepository.save(card);  
 return card.toString();  
 }  
  
 public List<Card> getCards() {  
 return cardRepository.findAll();  
 }  
  
 public void deleteById(Long cardId) {  
 cardRepository.deleteById(cardId);  
 }  
}

Листинг 21.2 – код класса CardService.

package ru.practic22;  
  
import jakarta.transaction.Transactional;  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
@RequiredArgsConstructor  
@Slf4j  
@Transactional  
public class BankService {  
 private final BankRepository bankRepository;  
 private final EmailService emailService;  
 public List<Bank> getBanks() {  
 log.info("finding all banks");  
 return bankRepository.findAll();  
 }  
  
 public Bank findByName(String name) {  
 log.info("finding bank by name");  
 return bankRepository.getBankByName(name).orElseThrow();  
 }  
  
 public Bank findById(Long id) {  
 log.info("finding bank by id");  
 return bankRepository.findById(id).orElseThrow();  
 }  
  
 public Bank saveBank(Bank bank) {  
 emailService.saveBank(bank);  
 log.info("saving bank");  
 return bankRepository.save(bank);  
 }  
  
 public void deleteById(Long id) {  
 log.info("deleting bank");  
 bankRepository.deleteById(id);  
 }  
}

Листинг 21.3 – код класса BankService.

spring.application.name=practic21  
spring.jpa.hibernate.ddl-auto=create-drop  
spring.datasource.driver-class-name=org.postgresql.Driver  
spring.jpa.show-sql=true  
spring.datasource.url=jdbc:postgresql://localhost/postgres  
spring.datasource.username=root  
spring.datasource.password=root  
logging.config=classpath:logback.xml  
  
spring.mail.host=smtp.gmail.com  
spring.mail.port=587  
spring.mail.username=\*\*\*\*  
spring.mail.password=\*\*\*\*  
spring.mail.properties.mail.smtp.auth=true  
spring.mail.properties.mail.smtp.starttls.enable=true

Листинг 21.4 – код файла application.properties.

## Тестирование.

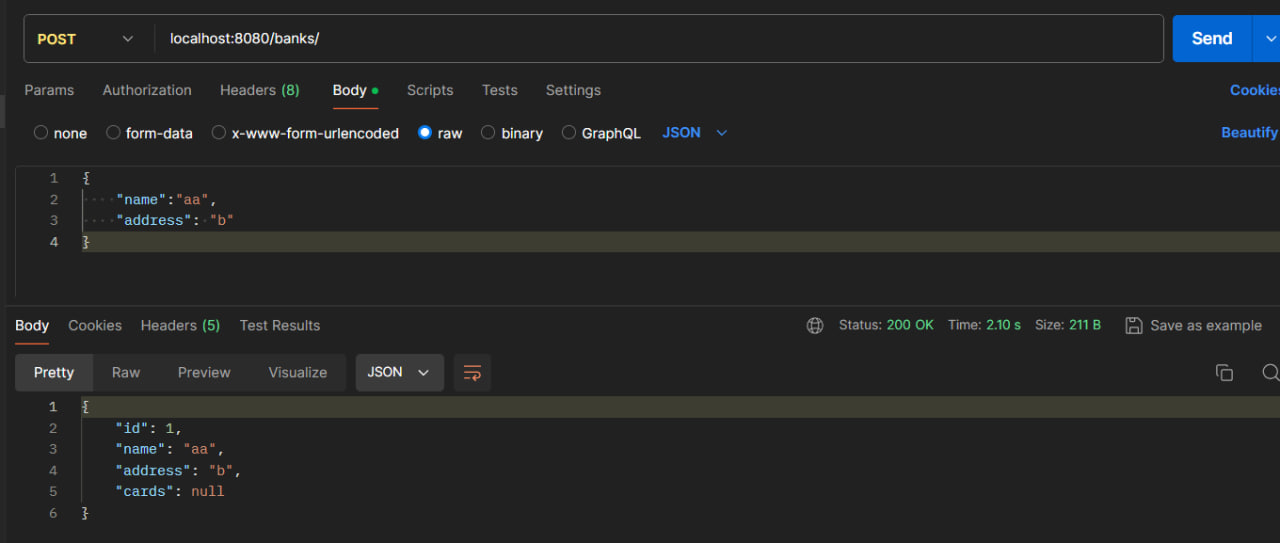


Рисунок 21.1 – результат работы программы.

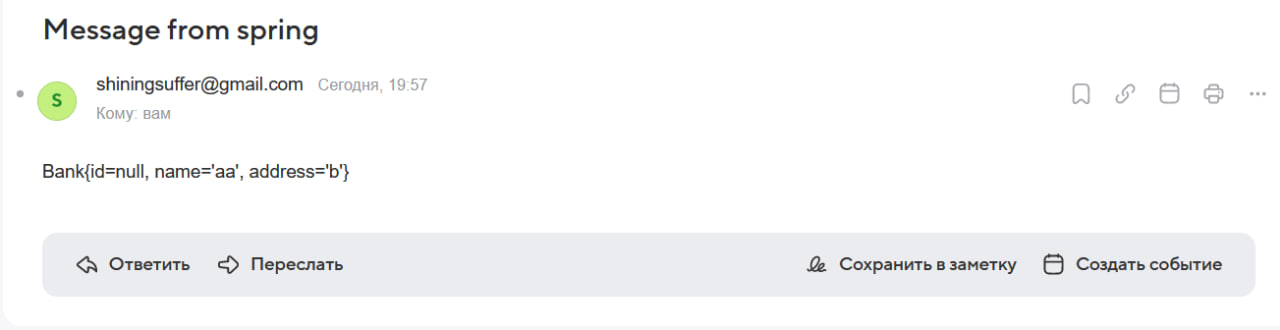


Рисунок 21.2 – результат работы программы.

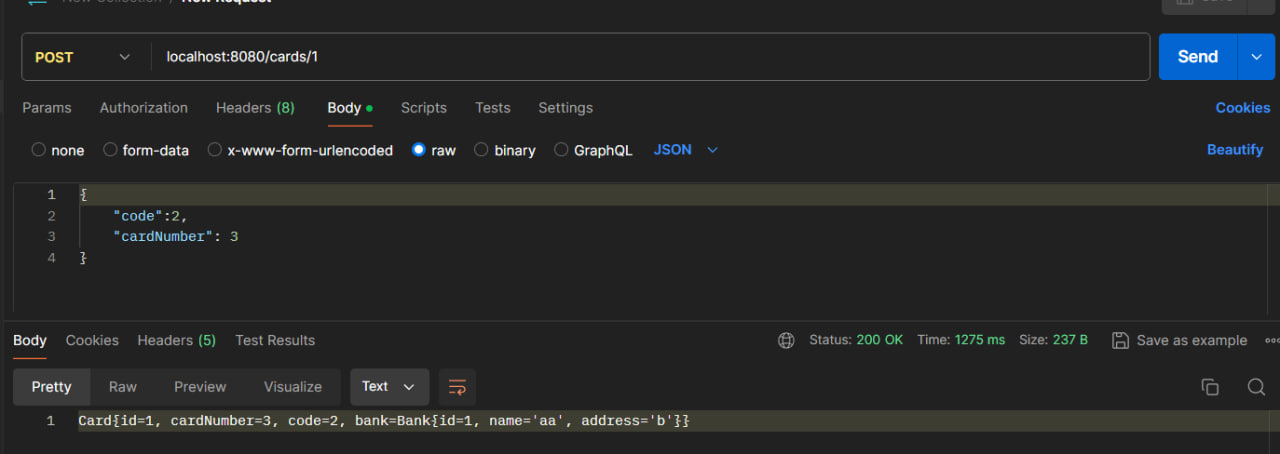


Рисунок 21.3 – результат работы программы.

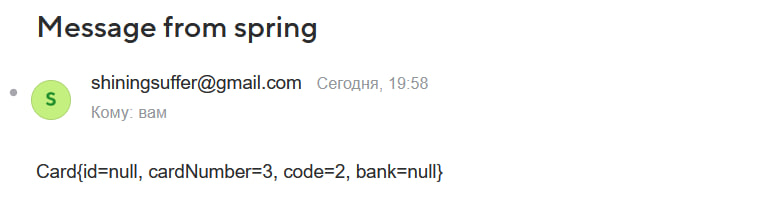


Рисунок 21.4 – результат работы программы.

## Вывод.

Поработал с проксированием, аннотацией Transactional, аннотацией Async.

# Практическая работа №22

## Цель работы.

Планирование заданий. Scheduler в Spring.

## Задание.

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

## Код программы.

package ru.practic22;  
  
import jakarta.annotation.PostConstruct;  
import lombok.AllArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.scheduling.annotation.EnableScheduling;  
import org.springframework.scheduling.annotation.Scheduled;  
import org.springframework.stereotype.Service;  
  
import javax.management.MBeanServer;  
import javax.management.ObjectName;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.lang.management.ManagementFactory;  
import java.nio.file.Files;  
import java.nio.file.Path;  
  
@Service  
@EnableScheduling  
@AllArgsConstructor  
@Slf4j  
public class ScheduledTask implements TaskMXBean {  
 private final BankService bankService;  
 private final CardService cardService;  
  
 @PostConstruct  
 private void init() throws Exception {  
 MBeanServer mbs = ManagementFactory.*getPlatformMBeanServer*();  
 ObjectName name = new ObjectName("ru.practic22:type=ScheduledTask");  
 if (!mbs.isRegistered(name)) {  
 mbs.registerMBean(this, name);  
 } else {  
 log.info("MBean with name {} is already registered.", name);  
 }  
 }  
  
  
 @Scheduled(fixedRateString = "PT30M")  
 public void remadeFiles() throws IOException {  
 Files.*walk*(Path.*of*("C:/study/jaba/practic22/src/main/resources/entities")).filter(Files::*isRegularFile*).forEach(p -> {  
 try {  
 log.info("deleting file in scheduled");  
 Files.*delete*(p);  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 });  
  
 FileWriter banksFile = new FileWriter("C:/study/jaba/practic22/src/main/resources/entities/banks.txt");  
 banksFile.write(bankService.getBanks().toString());  
  
 FileWriter cardsFile = new FileWriter("C:/study/jaba/practic22/src/main/resources/entities/cards.txt");  
 cardsFile.write(cardService.getCards().toString());  
 log.info("written to files in scheduled");  
 banksFile.close();  
 cardsFile.close();  
 }  
  
 @Override  
 public void runRemadeFiles() throws IOException {  
 remadeFiles();  
 }  
}

Листинг 22.1 – код класса ScheduledTask.

package ru.practic22;  
  
import java.io.IOException;  
  
public interface TaskMXBean {  
 void runRemadeFiles() throws IOException;  
}

Листинг 22.2 – код интерфейса TaskMXBean.

## Тестирование.

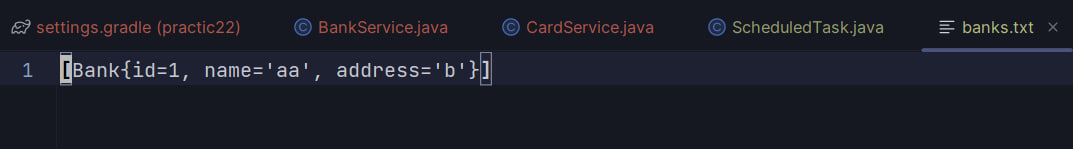


Рисунок 22 – результат работы программы.

## Вывод.

Получены навыки планирования заданий в Spring через Scheduler.

# Практическая работа №23

## Цель работы.

Использование Spring Security для аутентификации и авторизации пользователей.

## Задание.

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

## Код программы.

package ru.practic24;  
  
import jakarta.servlet.DispatcherType;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.config.Customizer;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.provisioning.InMemoryUserDetailsManager;  
import org.springframework.security.web.SecurityFilterChain;  
  
@Configuration  
@EnableWebSecurity  
public class SecurityConfig {  
 @Bean  
 public PasswordEncoder encoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
 @Bean  
 public InMemoryUserDetailsManager userDetailsService() {  
 return new InMemoryUserDetailsManager();  
 }  
  
 @Bean  
 public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
 http  
 .csrf(AbstractHttpConfigurer::disable)  
 .cors(AbstractHttpConfigurer::disable)  
 .authorizeHttpRequests(  
 (authorize) -> authorize  
 .dispatcherTypeMatchers(DispatcherType.FORWARD, DispatcherType.ERROR).permitAll()  
 .requestMatchers("/register").permitAll()  
 .requestMatchers("/\*\*").hasAuthority("SIMPLE\_USER")  
 .anyRequest().authenticated())  
 .httpBasic(Customizer.*withDefaults*())  
 .formLogin(Customizer.*withDefaults*());  
 return http.build();  
 }  
}

Листинг 23.1 – код класса SecurityConfig.

package ru.practic24;  
  
import lombok.Getter;  
import lombok.Setter;  
  
@Getter  
@Setter  
public class CustomUser {  
 private String name;  
 private String password;  
}

Листинг 23.2 – код класса CustomUser.

package ru.practic24;  
  
import lombok.AllArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.userdetails.User;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.provisioning.InMemoryUserDetailsManager;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.ModelAttribute;  
import org.springframework.web.bind.annotation.PostMapping;  
  
import java.util.List;  
  
@Controller  
@AllArgsConstructor  
@Slf4j  
public class AuthController {  
 private final InMemoryUserDetailsManager userDetailsManager;  
 private final PasswordEncoder passwordEncoder;  
  
 @GetMapping("/register")  
 public String showRegistrationForm(Model model) {  
 model.addAttribute("customUser", new CustomUser());  
 return "registration-form";  
 }  
  
 @PostMapping("/register")  
 public String registerUser(@ModelAttribute("customUser") CustomUser user) {  
 if (userDetailsManager.userExists(user.getName())) {  
 log.info("!!!! ERRRROOOOORRR !!!!");  
 }  
 String encodedPassword = passwordEncoder.encode(user.getPassword());  
 userDetailsManager.createUser(new User(user.getName(), encodedPassword, List.*of*(new SimpleGrantedAuthority("SIMPLE\_USER"))));  
 return "redirect:/login";  
 }  
}

Листинг 23.3 – код класса AuthController.

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Registration Form</title>  
</head>  
<body>  
<h2>Registration Form</h2>  
<form action="/register" th:action="@{/register}" th:object="${customUser}" method="post">  
 <label>  
 <input type="text" th:field="\*{name}" placeholder = "name"/>  
 </label>  
 <label>  
 <input type="text" th:field="\*{password}" placeholder = "password"/>  
 </label>  
 <p><input type="submit" value="Submit" />  
</form>  
</body>  
</html>

Листинг 23.4 – код файла register-form.html.

## Тестирование.

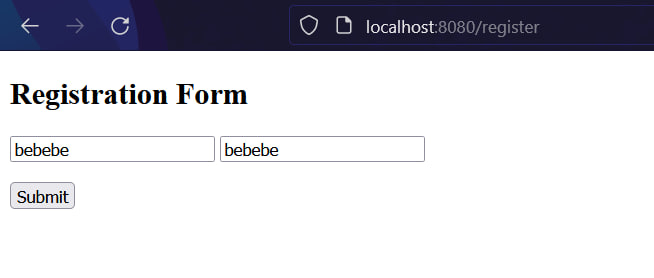


Рисунок 23.1 – результат работы программы.

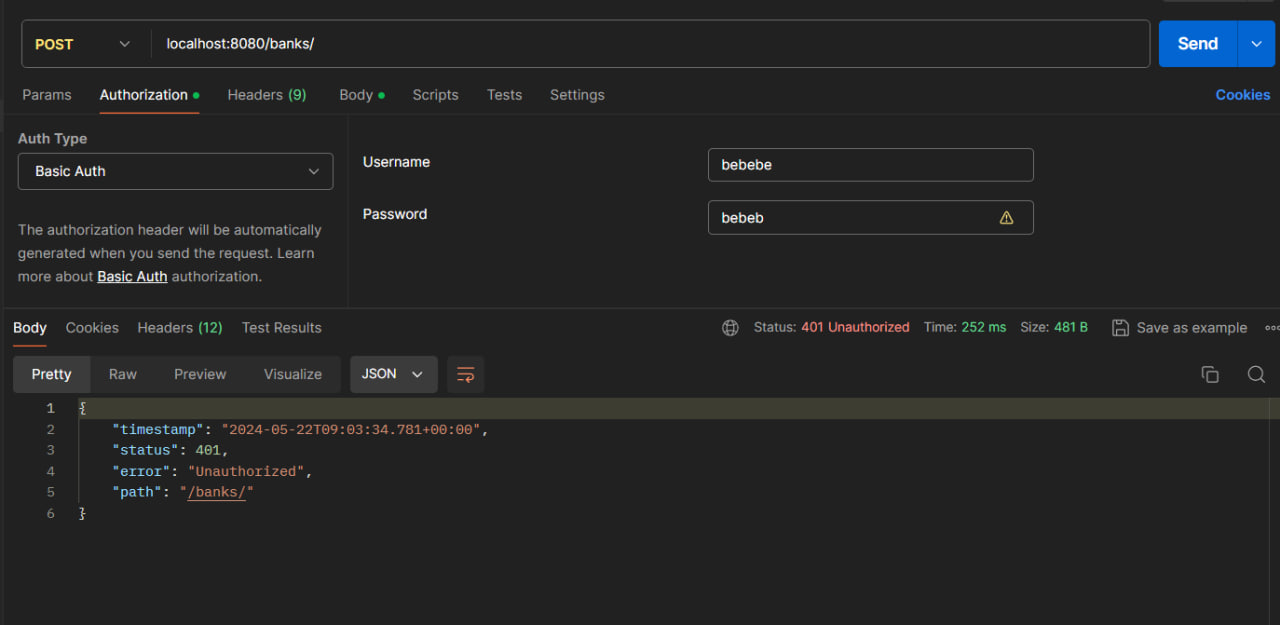


Рисунок 23.2 – результат работы программы.

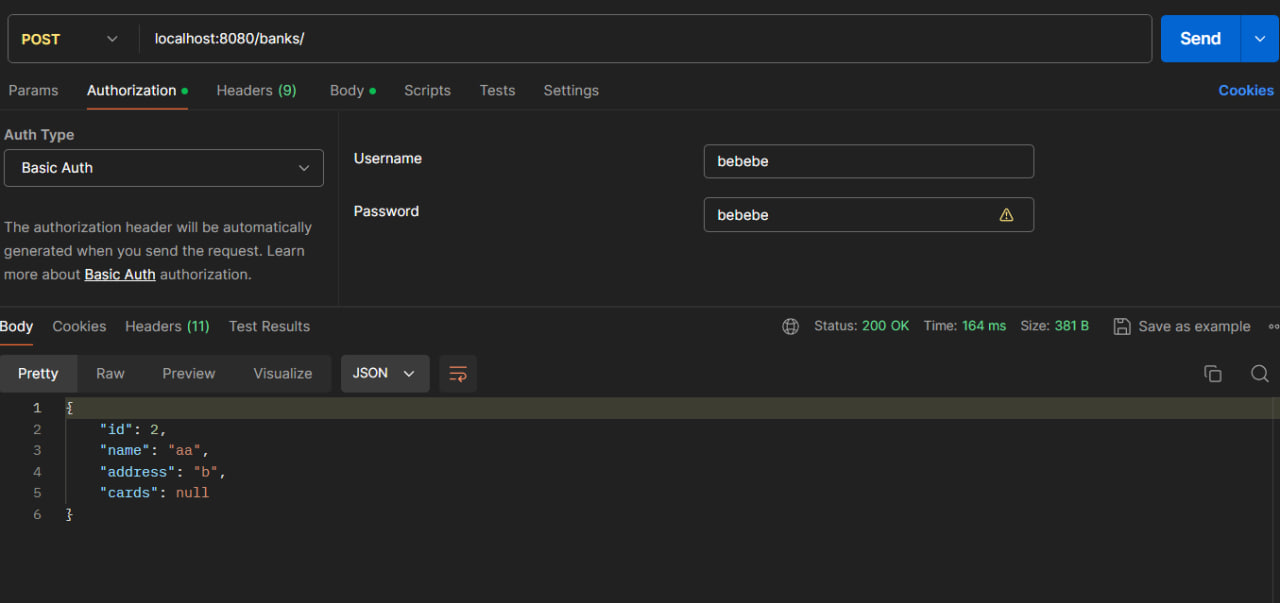


Рисунок 23.3 – результат работы программы.

## Вывод.

Научился пользоваться Spring Security для аутентификации и авторизации пользователей.

# Практическая работа №24

## Цель работы.

Тестирование в Spring Framework с использованием Junit.

## Задание.

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

## Код программы.

package ru.practic24;  
  
import org.junit.jupiter.api.Test;  
import org.springframework.boot.test.context.SpringBootTest;  
  
@SpringBootTest  
class Practic24Test {  
  
 @Test  
 void contextLoads() {  
 }  
  
}

Листинг 24.1 – код класса Practic24Test.

package ru.practic24;  
  
  
import org.junit.jupiter.api.Assertions;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.Mockito;  
import org.mockito.junit.jupiter.MockitoExtension;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
  
@ExtendWith(MockitoExtension.class)  
public class ServicesTest {  
 @Mock  
 private BankRepository bankRepository;  
 @Mock  
 private CardRepository cardRepository;  
 @InjectMocks  
 private BankService bankService;  
 @InjectMocks  
 private CardService cardService;  
  
 @Test  
 public void getBanks() {  
 List<Bank> banks = new ArrayList<>();  
 banks.add(new Bank(Long.*valueOf*(1), "bank1", "road1", null));  
 banks.add(new Bank(Long.*valueOf*(2), "bank2", "road2", null));  
  
 Mockito.*when*(bankRepository.findAll()).thenReturn(banks);  
  
 Iterable<Bank> bankIterable = bankService.getBanks();  
 List<Bank> bankList = new ArrayList<>();  
 bankIterable.forEach(bankList::add);  
 *assertEquals*(2, bankList.size());  
 }  
  
 @Test  
 public void addBank() {  
 Bank bank = new Bank(Long.*valueOf*(2), "bank2", "road2", null);  
 Mockito.*when*(bankRepository.save(bank)).thenReturn(bank);  
  
 *assertEquals*(bank, bankService.saveBank(bank));  
 }  
  
 @Test  
 public void deleteBank() {  
 long bankId = 1;  
  
 Mockito.*doNothing*().when(bankRepository).deleteById(bankId);  
  
 Assertions.*assertDoesNotThrow*(() -> bankService.deleteById(bankId));  
 }  
  
 @Test  
 public void getBankByName() {  
 String bankName = "Bank1";  
 Bank bank = new Bank(Long.*valueOf*(1), bankName, "road1", null);  
  
  
 Mockito.*when*(bankRepository.getBankByName(bankName)).thenReturn(Optional.*of*(bank));  
  
 Bank result = bankService.findByName(bankName);  
  
 *assertEquals*(bank, result);  
 }  
  
 @Test  
 public void getCards() {  
 List<Card> cards = new ArrayList<>();  
 cards.add(new Card(Long.*valueOf*(1), 1, 1, null));  
 cards.add(new Card(Long.*valueOf*(2), 2, 2, null));  
  
 Mockito.*when*(cardRepository.findAll()).thenReturn(cards);  
  
 Iterable<Card> cardsIterable = cardService.getCards();  
 List<Card> cardsList = new ArrayList<>();  
 cardsIterable.forEach(cardsList::add);  
 *assertEquals*(2, cardsList.size());  
 }  
  
 @Test  
 public void addCard() {  
 Card card = new Card(Long.*valueOf*(1), 1, 1, null);  
  
 Mockito.*when*(cardRepository.save(card)).thenReturn(card);  
  
 *assertEquals*(card.toString(), cardService.saveCard(card, Long.*valueOf*(0)));  
 }  
  
 @Test  
 public void deleteSubject() {  
 long cardId = 1;  
  
 Mockito.*doNothing*().when(cardRepository).deleteById(cardId);  
  
 Assertions.*assertDoesNotThrow*(() -> cardService.deleteById(cardId));  
 }  
  
  
}

Листинг 24.2 – код класса ServicesTest.

## Тестирование.

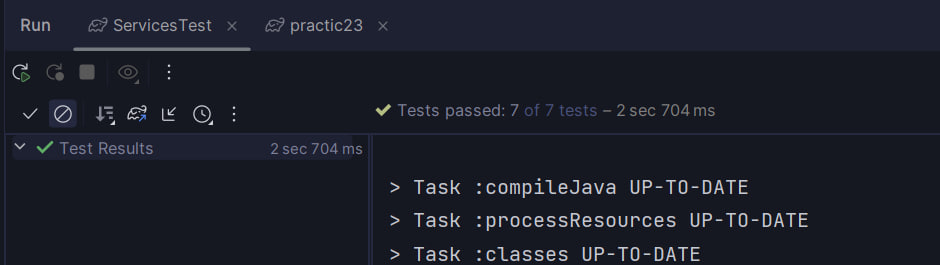


Рисунок 24 – результат работы программы.

## Вывод.

Ознакомились с тестированием в Spring Framework с использованием Junit.

**СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ**

1. Стелтинг С., Маасен О. Применение шаблонов Java. Библиотека профессионала.: Пер. с англ. — М.: Издательский дом "Вильяме", 2002. — 576 с.: ил. — Парал. тит. англ.

2. Functional Interfaces in Java: Fundamentals and Examples 1st ed. Edition, Kindle Edition [Электронный ресурс]. URL: https://www.amazon.com/Functional-Interfaces-Java-Fundamentals-Examples-ebook/dp/B07NRHQSCW (дата обращения: 29.01.21). Заголовок с экрана.

3. Hibernate Search 6.0.0.Final: Reference Documentation [Электронный ресурс]. URL: https://docs.jboss.org/hibernate/stable/search/reference/en-US/html\_single/ (дата обращения: 29.01.21). Заголовок с экрана.

4. Паттерны проектирования на Java. Каталог Java-примеров. [Электронный ресурс]. URL: https://refactoring.guru/ru/design-patterns/java (дата обращения: 29.01.21). Заголовок с экрана.

5. Руководство по Spring [Электронный ресурс]. URL: https://proselyte.net/tutorials/spring-tutorial-full-version/ (дата обращения: 29.01.21). Заголовок с экрана.

6. The Reactive Manifesto [Электронный ресурс]. URL: https://www.reactivemanifesto.org/ (дата обращения: 29.01.21). Заголовок с экрана.

7. Spring Framework Documentation [Электронный ресурс]. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/web.html (дата обращения: 29.01.21). Заголовок с экрана.

8. Hibernate Search 6.0.0. Final: Reference Documentation [Электронный ресурс]. URL: https://docs.jboss.org/hibernate/stable/search/reference/en-US/html\_single/ (дата обращения: 29.01.21). Заголовок с экрана.