



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

« 30 » мая
2024г.

Наместников В.Н.

(подпись)

Принял:

Преподаватель кафедры ИиППО
ИИТ

« 30 » мая
2024г.

Ермаков С.Р.

(подпись)

Москва 2023 г.

Содержание

1 Практическая работа №1	7
1.1 Цель работы.....	7
1.2 Задание.....	7
1.3 Тестирование.....	7
1.4 Выводы.....	7
2 Практическая работа №2.....	8
2.1 Цель работы.....	8
2.2 Задание.....	8
2.3 Код программы.....	8
2.4 Тестирование.....	10
2.5 Вывод.....	10
3 Практическая работа №3.....	11
3.1 Цель работы.....	11
3.2 Задание.....	11
3.3 Код программы.....	11
3.4 Тестирование.....	16
3.5 Вывод.....	16
4 Практическая работа №4.....	17
4.1 Цель работы.....	17
4.2 Задание.....	17
4.3 Код программы.....	17
4.4 Тестирование.....	19
4.5 Вывод.....	19
5 Практическая работа №5.....	20
5.1 Цель работы.....	20
5.2 Задание.....	20
5.3 Код программы.....	20
5.4 Тестирование.....	21
5.5 Вывод.....	21

6 Практическая работа №6.....	22
6.1 Цель работы.....	22
6.2 Задание.....	22
6.3 Код программы.....	22
6.4 Тестирование.....	23
6.5 Вывод.....	23
7 Практическая работа №7.....	24
7.1 Цель.....	24
7.2 Задание.....	24
7.3 Код программы.....	24
7.4 Тестирование.....	27
7.5 Вывод.....	27
8 Практическая работа №8.....	28
8.1 Цель работы.....	28
8.2 Задание.....	28
8.3 Код программы.....	28
8.4 Тестирование.....	31
8.5 Вывод.....	31
9 Практические работы №9.....	32
9.1 Цель работы.....	32
9.2 Задание.....	32
9.3 Тестирование.....	33
9.4 Вывод.....	33
10 Практическая работа №10.....	34
10.1 Цель работы.....	34
10.2 Задание.....	34
10.3 Код программы.....	34
10.4 Тестирование.....	35
10.5 Вывод.....	35
11 Практическое задание №11.....	36

11.1 Цель работы.....	36
11.2 Задание.....	36
11.3 Код программы.....	36
11.4 Тестирование.....	37
11.5 Вывод.....	37
12 Практическая работа №12.....	38
12.1 Цель работы.....	38
12.2 Задание.....	38
12.3 Код программы.....	38
12.4 Тестирование.....	39
12.5 Вывод.....	39
13 Практическая работа №13.....	40
13.1 Цель работы.....	40
13.2 Задание.....	40
13.3 Код программы.....	40
13.4 Тестирование.....	41
13.5 Вывод.....	41
14 Практическая работа №14.....	42
14.1 Цель работы.....	42
14.2 Задание.....	42
14.3 Код программы.....	42
14.4 Тестирование.....	45
14.5 Вывод.....	45
15 Практическая работа №15.....	46
15.1 Цель работы.....	46
15.2 Задание.....	46
15.3 Код программы.....	46
15.4 Тестирование.....	48
15.5 Вывод.....	48
16 Практическая работа №16.....	49

16.1 Цель работы.....	49
16.2 Задание.....	49
16.3 Код программы.....	49
16.4 Тестирование.....	52
16.5 Вывод.....	52
17 Практическая работа №17.....	53
17.1 Цель работы.....	53
17.2 Задание.....	53
17.3 Код программы.....	53
17.4 Тестирование.....	54
17.5 Вывод.....	54
18 Практическая работа №18.....	55
18.1 Цель работы.....	55
18.2 Задание.....	55
18.3 Код программы.....	55
18.4 Тестирование.....	58
18.5 Вывод.....	58
19 Практическая работа №19.....	59
19.1 Цель работы.....	59
19.2 Задание.....	59
19.3 Код программы.....	59
19.4 Тестирование.....	61
19.5 Вывод.....	61
20 Практическая работа №20.....	62
20.1 Цель работы.....	62
20.2 Задание.....	62
20.3 Код программы.....	62
20.4 Тестирование.....	63
20.5 Вывод.....	63
21 Практическая работа №21.....	64

21.1 Цель работы.....	64
21.2 Задание.....	64
21.3 Код программы.....	64
21.4 Тестирование.....	67
21.5 Вывод.....	68
22 Практическая работа №22.....	69
22.1 Цель работы.....	69
22.2 Задание.....	69
22.3 Код программы.....	69
22.4 Тестирование.....	71
22.5 Вывод.....	71
23 Практическая работа №23.....	72
23.1 Цель работы.....	72
23.2 Задание.....	72
23.3 Код программы.....	72
23.4 Тестирование.....	75
23.5 Вывод.....	76
24 Практическая работа №24.....	77
24.1 Цель работы.....	77
24.2 Задание.....	77
24.3 Код программы.....	77
24.4 Тестирование.....	79
24.5 Вывод.....	80
25 Список использованных источников.....	81

1 ПРАКТИЧЕСКАЯ РАБОТА №1

1.1 Цель работы.

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

1.2 Задание.

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

Код программы.

```
import java.util.Arrays;
import java.util.function.Function;

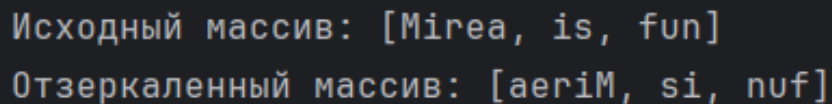
public class MirrorStrings {
    public static void main(String[] args) {
        Function<String[], String[]> mirrorFunction = strings ->
            Arrays.stream(strings)
                .map(str -> new StringBuilder(str).reverse().toString())
                .toArray(String[]::new);

        String[] inputArray = {"Mirea", "is", "fun"};
        String[] mirroredArray = mirrorFunction.apply(inputArray);

        System.out.println("Исходный массив: " + Arrays.toString(inputArray));
        System.out.println("Отзеркаленный массив: " + Arrays.toString(mirroredArray));
    }
}
```

Листинг 1.1 – класс Student.

1.3 Тестирование.



```
Исходный массив: [Mirea, is, fun]
Отзеркаленный массив: [aeriM, si, nuf]
```

Рисунок 1 - результат работы программы.

1.4 Выводы.

Освоили работу с функциональными интерфейсами в Java.

2 ПРАКТИЧЕСКАЯ РАБОТА №2

2.1 Цель работы.

Работа со Stream API в Java 8.

2.2 Задание.

Сортировка по весу в обратном порядке, фильтрация по фамилии не Иванов, сортировка по возрасту, произведение всех возрастов

2.3 Код программы.

```
package pra2;

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 (
            firstName + " " +
            lastName + " " +
            age + " years, " +
            weight + " kg"
        );
    }
}
```

Листинг 2.1 – код класса Human.


```

import java.util.List;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.Comparator;
import java.util.stream.Collectors;

public class Main {
    public static void main(String[] args) {
        List<Human> humans = List.of(
            new Human(19, "Vladimir", "Namestnikov", LocalDate.parse("14-08-2004",
DateTimeFormatter.ofPattern("dd-MM-yyyy")), 90),
            new Human(22, "Dilyra", "Sultonova", LocalDate.parse("26-12-2001",
DateTimeFormatter.ofPattern("dd-MM-yyyy")), 55),
            new Human(27, "Vladislav", "Namestnikov", LocalDate.parse("08-06-1996",
DateTimeFormatter.ofPattern("dd-MM-yyyy")), 90),
            new Human(18, "Ivan", "Ivanov", LocalDate.parse("01-01-2005",
DateTimeFormatter.ofPattern("dd-MM-yyyy")), 72)
        );

        System.out.println("All humans");
        humans.forEach(System.out::println);

        List<Human> sortedWeight = humans.stream().sorted(Comparator.comparingInt(h -> -
h.weight)).collect(Collectors.toList());
        System.out.println("\nСписок людей, отсортированный по весу в обратном
порядке:");
        sortedWeight.forEach(System.out::println);

        List<Human> neIvanov = humans.stream().filter(h -> h.lastName !=
"Ivanov").collect(Collectors.toList());
        System.out.println("\nСписок людей не с фамилией Иванов:");
        neIvanov.forEach(System.out::println);

        List<Human> sortedAge = humans.stream().sorted(Comparator.comparingInt(h ->
h.age)).collect(Collectors.toList());
        System.out.println("\nСписок людей, отсортированный по возрасту:");
        sortedAge.forEach(System.out::println);

        int prodAges = humans.stream().mapToInt(h -> h.age).reduce(1, (a, b) -> (a * b));
        System.out.println("\nПроизведение всех возрастов: ");
        System.out.println(prodAges);
    }
}

```

Листинг 2.2 – код класса Main.

2.4 Тестирование.

```
All humans
Vladimir Namestnikov 19 years, 90 kg
Dilyra Sultonova 22 years, 55 kg
Vladislav Namestnikov 27 years, 90 kg
Ivan Ivanov 18 years, 72 kg

Список людей, отсортированный по весу в обратном порядке:
Vladimir Namestnikov 19 years, 90 kg
Vladislav Namestnikov 27 years, 90 kg
Ivan Ivanov 18 years, 72 kg
Dilyra Sultonova 22 years, 55 kg

Список людей не с фамилией Иванов:
Vladimir Namestnikov 19 years, 90 kg
Dilyra Sultonova 22 years, 55 kg
Vladislav Namestnikov 27 years, 90 kg

Список людей, отсортированный по возрасту:
Ivan Ivanov 18 years, 72 kg
Vladimir Namestnikov 19 years, 90 kg
Dilyra Sultonova 22 years, 55 kg
Vladislav Namestnikov 27 years, 90 kg

Произведение всех возрастов:
203148
```

Рисунок 2 - результат работы программы.

2.5 Вывод.

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

3 ПРАКТИЧЕСКАЯ РАБОТА №3

3.1 Цель работы.

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

3.2 Задание.

Реализовать потокобезопасную коллекцию `Map` с использованием `Lock`, а также потокобезопасный `Set` с использованием ключевого слова `synchronized`.

3.3 Код программы.

```
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 SafeMap<K, V> implements Map<K, V> {
    private final Map<K, V> map = new HashMap<>();
    private final Lock lock = new ReentrantLock();

    @Override
    public int size() {
        lock.lock();
        try {
            return map.size();
        } finally {
            lock.unlock();
        }
    }

    @Override
    public boolean isEmpty() {
        lock.lock();
        try {
            return map.isEmpty();
        } finally {
```

```

        lock.unlock();
    }
}

@Override
public boolean containsKey(Object key) {
    lock.lock();
    try {
        return map.containsKey(key);
    } finally {
        lock.unlock();
    }
}

@Override
public boolean containsValue(Object value) {
    lock.lock();
    try {
        return map.containsValue(value);
    } finally {
        lock.unlock();
    }
}

@Override
public V get(Object key) {
    lock.lock();
    try {
        return map.get(key);
    } finally {
        lock.unlock();
    }
}

@Override
public V put(K key, V value) {
    lock.lock();
    try {
        return map.put(key, value);
    } finally {
        lock.unlock();
    }
}

@Override

```

```

public V remove(Object key) {
    lock.lock();
    try {
        return map.remove(key);
    } finally {
        lock.unlock();
    }
}

@Override
public void putAll(Map<? extends K, ? extends V> m) {
    lock.lock();
    try {
        map.putAll(m);
    } finally {
        lock.unlock();
    }
}

@Override
public void clear() {
    lock.lock();
    try {
        map.clear();
    } finally {
        lock.unlock();
    }
}

@Override
public Set<K> keySet() {
    lock.lock();
    try {
        return map.keySet();
    } finally {
        lock.unlock();
    }
}

@Override
public Collection<V> values() {
    lock.lock();
    try {
        return map.values();
    } finally {

```

```

        lock.unlock();
    }
}

@Override
public Set<Entry<K, V>> entrySet() {
    return null;
}
}

```

Листинг 3.1 – код класса SafeMap.

```

import java.util.Collection;
import java.util.Iterator;
import java.util.Set;
import java.util.HashSet;

public class SafeSet<T> implements Set<T> {
    private final Set<T> 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<T> iterator() {
        return set.iterator();
    }

    @Override
    public synchronized Object[] toArray() {
        return set.toArray();
    }
}

```

```

@Override
public synchronized <T1> T1[] toArray(T1[] a) {
    return set.toArray(a);
}

@Override
public synchronized boolean add(T t) {
    return set.add(t);
}

@Override
public synchronized boolean remove(Object o) {
    return set.remove(o);
}

@Override
public synchronized boolean containsAll(Collection<?> c) {
    return set.containsAll(c);
}

@Override
public synchronized boolean addAll(Collection<? extends T> c) {
    return set.addAll(c);
}

@Override
public synchronized boolean retainAll(Collection<?> c) {
    return set.retainAll(c);
}

@Override
public synchronized boolean removeAll(Collection<?> c) {
    return set.removeAll(c);
}

@Override
public synchronized void clear() {
    set.clear();
}
}

```

Листинг 3.2 – код класса SafeSet.

3.4 Тестирование.

```
Добавление в множество: 3, Размер: 4
Добавление в множество: 1, Размер: 2
Добавление в множество: 2, Размер: 3
Добавление в множество: 0, Размер: 1
Добавление в множество: 4, Размер: 5
Конечный размер множества: 5
Добавление в словарь: Ключ: 1, Значение: 1, размер: 1
Добавление в словарь: Ключ: 0, Значение: 0, размер: 4
Добавление в словарь: Ключ: 3, Значение: 3, размер: 2
Добавление в словарь: Ключ: 2, Значение: 2, размер: 5
Добавление в словарь: Ключ: 4, Значение: 4, размер: 3
Конечный размер словаря: 5
```

Рисунок 3 - результат работы SafeMap и SafeSet.

3.5 Вывод.

Изучены основы работы с потоками. Реализованы потокобезопасные коллекции.

4 ПРАКТИЧЕСКАЯ РАБОТА №4

4.1 Цель работы.

Работа с ExecutorService, CompletableFuture.

4.2 Задание.

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

4.3 Код программы.

```
import java.util.LinkedList;
import java.util.List;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.Executor;
import java.util.concurrent.LinkedBlockingQueue;

public class ExecutorService implements Executor {
    private final BlockingQueue<Runnable> taskQueue;
    private final List<WorkerThread> threads;

    public ExecutorService(int numThreads) {
        this.taskQueue = new LinkedBlockingQueue<>();
        this.threads = new LinkedList<>();

        for (int i = 0; i < numThreads; i++) {
            WorkerThread thread = new WorkerThread();
            thread.start();
            threads.add(thread);
        }
    }

    @Override
    public void execute(Runnable command) {
        try {
            taskQueue.put(command);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }

    private class WorkerThread extends Thread {
        @Override
        public void run() {
            while (true) {
```

```

        try {
            Runnable task = taskQueue.take();
            task.run();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            break;
        }
    }
}

public void shutdown() {
    for (WorkerThread thread: threads) {
        thread.interrupt();
    }
}
}

```

Листинг 4.1 – код класса ExecutorService.

```

public class Main {
    public static void main(String[] args) {
        ExecutorService executorService = new ExecutorService(2);

        executorService.execute(() -> {
            int x = 0;
            while (x < 14)
                System.out.println(x++);
        });

        executorService.execute(() -> {
            System.out.println("Start");
        });
    }
}

```

Листинг 4.2 – код класса Main.

4.4 Тестирование.

A screenshot of a terminal window with a dark background. The text is displayed in a light gray, monospaced font. It starts with the word "Start" on the first line, followed by a vertical list of integers from 0 to 13, one per line.

```
Start
0
1
2
3
4
5
6
7
8
9
10
11
12
13
```

Рисунок 4 – результат тестирования программы.

4.5 Вывод.

Изучена работа с `ExecutorService`, `CompletableFuture`.

5 ПРАКТИЧЕСКАЯ РАБОТА №5

5.1 Цель работы.

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

5.2 Задание.

Реализовать паттерн Singleton как минимум 3-мя способами.

5.3 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.

5.4 Тестирование.

```

Message from singleton 1
Message from singleton2
Message from singleton 3

```

Рисунок 5 – результат тестирования программы.

5.5 Вывод.

Освоен паттерн Singleton.

6 ПРАКТИЧЕСКАЯ РАБОТА №6

6.1 Цель работы.

Знакомство с реализацией порождающих паттернов проектирования.

6.2 Задание.

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

6.3 Код программы.

```
public interface Prototype {  
    Prototype clone();  
}
```

Листинг 6.1 – код интерфейса «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.

```
public class Main {  
    public static void main(String[] args) {  
  
        Creator creator = new ConcreteCreator();  
        Product product = creator.create();  
        product.perform();  
  
        CarFactory fabric = new MercedesFactory();  
        fabric.createCoupe().drive();  
        fabric.createSedan().drive();  
    }  
}
```

```

    fabric = new LadaFactory();
    fabric.createCoupe().drive();
    fabric.createSedan().drive();

    var point = new Director(new ConcreteBuilder()).createPoint();
    System.out.println(point);

    var dog1 = new Dog("Alice", 12);
    dog1.printInfo();
    var clone = dog1.clone();
    clone.printInfo();
}
}

```

Листинг 6.3 – код класса Main.

6.4 Тестирование.

```

Concrete product created by fabric method
Mercedes coupe drive
Mercedes sedan drive
Lada coupe drive
Lada sedan drive
Point{x=5, y=4, speed=10}
Dog's name is Alice
Dog's age is 12
Dog's name is Alice
Dog's age is 12

```

Рисунок 6 – результат тестирования программы.

6.5 Вывод.

Реализованы основные порождающие паттерны проектирования.

7 ПРАКТИЧЕСКАЯ РАБОТА №7

7.1 Цель.

Реализация структурных паттернов проектирования.

7.2 Задание.

Реализовать паттерны «Фасад» и «Заместитель».

7.3 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.

```

package proxy;

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();

var proxy = new CachingProxy();
System.out.println(proxy.getAnswer("Vanya"));
System.out.println(proxy.getAnswer("Vanya"));
}
```

Листинг 7.8 – код класса Main.

7.4 Тестирование.

```
Starting database
Starting http server
Started frontend server
Shutdown database
Shutdown http server
Shutdown frontend
Hello Vanya
Hello Vanya
```

Рисунок 7 – результат тестирования программы.

7.5 Вывод.

Изучены основные структурные паттерны проектирования.

8 ПРАКТИЧЕСКАЯ РАБОТА №8

8.1 Цель работы.

Реализация поведенческих паттернов проектирования.

8.2 Задание.

Реализовать паттерны «Посредник» и «Итератор».

8.3 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();

        MessageBroker msgBroker = new MessageBroker();
        Teacher u1 = new Teacher(msgBroker, "Vova");
        Student u2 = new Student(msgBroker, "Vlad");
        Student u3 = new Student(msgBroker, "Dilya");
        Teacher u4 = new Teacher(msgBroker, "Natasha");
        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.

8.4 Тестирование.

```

1
2
3
4
5
User Vova has received message: Message from User with name: Vova from Teacher
User Vlad has received message: Message from User with name: Vova from Teacher
User Dilya has received message: Message from User with name: Vova from Teacher
User Natasha has received message: Message from User with name: Vova from Teacher

```

Рисунок 8 – результат тестирования программы.

8.5 Вывод.

Изучены основные поведенческие паттерны проектирования.

9 ПРАКТИЧЕСКИЕ РАБОТЫ №9

9.1 Цель работы.

Знакомство с системой сборки Gradle.

9.2 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.

9.3 Тестирование.



Рисунок 9 – результат тестирования программы.

9.4 Вывод.

Изучена система сборки Gradle.

10 ПРАКТИЧЕСКАЯ РАБОТА №10

10.1 Цель работы.

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

10.2 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

10.3 Код программы.

```
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.

10.4 Тестирование.

```

2024-05-20T20:35:59.502+03:00
2024-05-20T20:35:59.505+03:00
2024-05-20T20:36:00.161+03:00
Junior Impl
Middle Impl
Senior Impl

```

Рисунок 10 – результат тестирования программы.

10.5 Вывод.

Изучены основные концепты работы с фреймворком Spring.

11 ПРАКТИЧЕСКОЕ ЗАДАНИЕ №11

11.1 Цель работы.

Разобраться с использованием Spring boot.

11.2 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

11.3 Код программы.

```
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.

11.4 Тестирование.

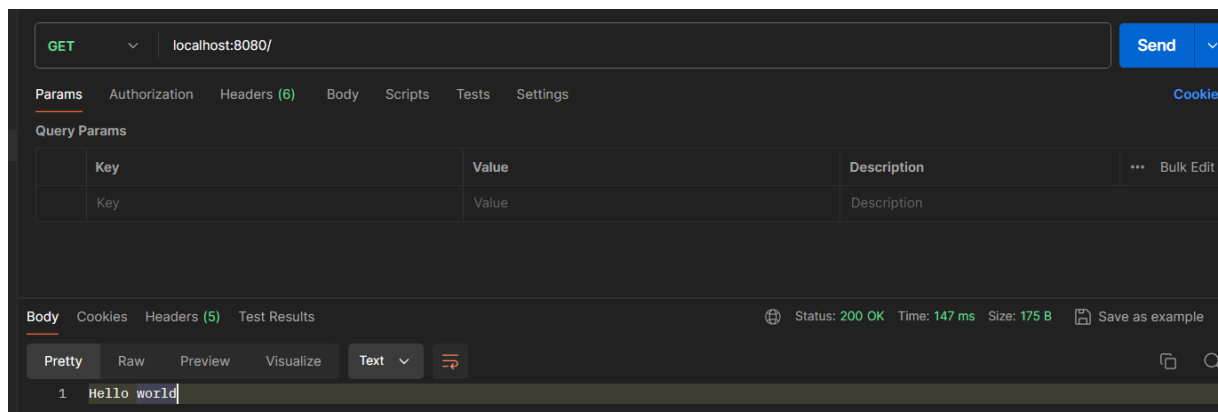


Рисунок 11.1 – результат тестирования программы.

11.5 Вывод.

Разобрались с использованием Spring boot.

12 ПРАКТИЧЕСКАЯ РАБОТА №12

12.1 Цель работы.

Работа с жизненным циклом компонентов. Аннотации PostConstruct, PreDestroy.

12.2 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.

12.3 Код программы.

```
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));
```

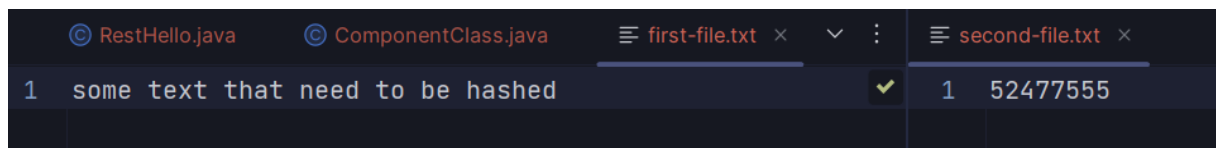
```

StringBuffer text = new StringBuffer();
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.

12.4 Тестирование.



© RestHello.java	© ComponentClass.java	≡ first-file.txt ×	⌵	⋮	≡ second-file.txt ×
1	some text that need to be hashed	✓	1	52477555	

Рисунок 12 – результат тестирования программы.

12.5 Вывод.

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

13 ПРАКТИЧЕСКАЯ РАБОТА №13

13.1 Цель работы.

Конфигурирование приложения. Environment.

13.2 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.

13.3 Код программы.

```
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: Vladimir
  last_name: Namestnikov
  group: IKBO-20-22
```

Листинг 13.2 – код файла application.yml.

13.4 Тестирование.

```
+03:00 INFO 18972 --- [main] ru.example.Main : Starting Main using Java 19.0.2 with PID 18972 (C:\D\*****\****\*****\*)
+03:00 INFO 18972 --- [main] ru.example.Main : No active profile set, falling back to 1 default profile: "default"
+03:00 INFO 18972 --- [main] ru.example.User : create User: User(name=Vladimir, last_name=Nesternikov, group=IKBO-20-22)
+03:00 INFO 18972 --- [main] ru.example.Main : Created main in 0.793 seconds (process running for 1.095)
```

Рисунок 13.1 – результат тестирования программы.

13.5 Вывод.

Разобрался с конфигурированием приложения, Environment.

14 ПРАКТИЧЕСКАЯ РАБОТА №14

14.1 Цель работы.

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

14.2 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.

14.3 Код программы.

```
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

```
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.

```

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.

```

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.

```

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.

14.4 Тестирование.



Наместников Владимир Николаевич ИКБО-20-22

Рисунок 14.1 – результат тестирования программы.

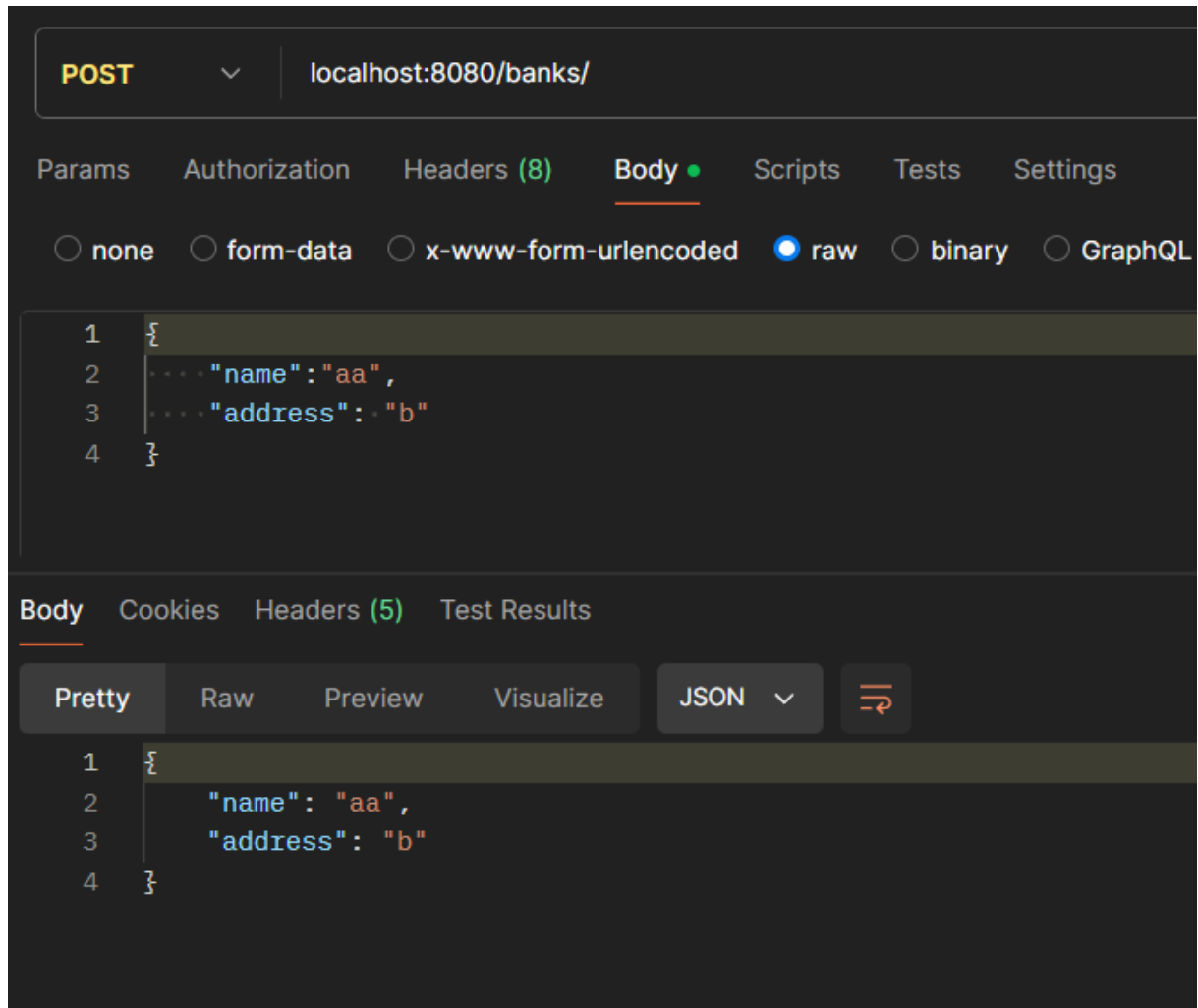


Рисунок 14.2 – результат тестирования программы.

14.5 Вывод.

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

15 ПРАКТИЧЕСКАЯ РАБОТА №15

15.1 Цель работы.

Использование Hibernate в Spring framework.

15.2 Задание.

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

15.3 Код программы.

```
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.

```
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.

```

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.

15.4 Тестирование.

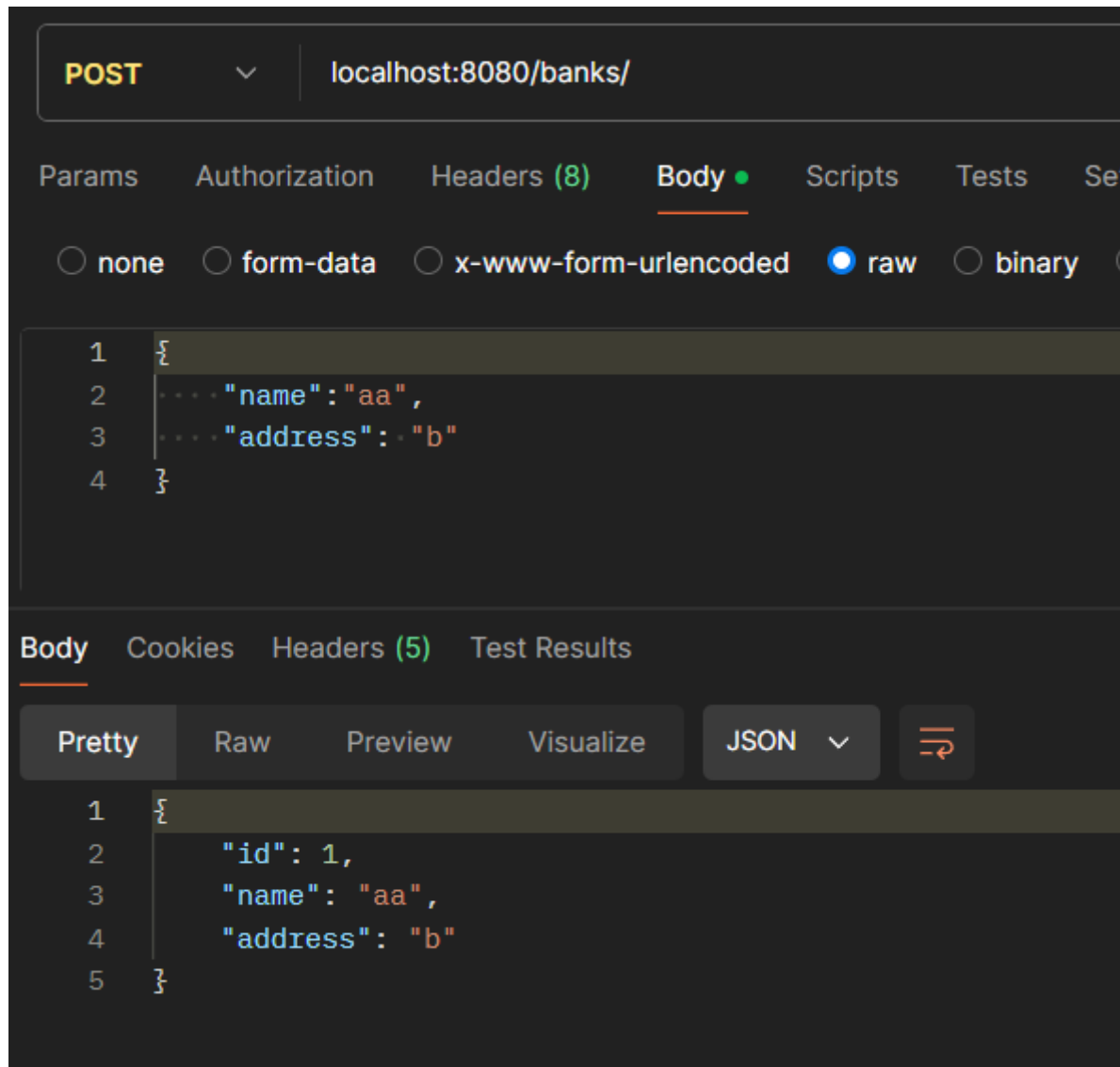


Рисунок 15 – результат тестирования программы.

15.5 Вывод.

Объекты сохраняются в базе данных PostgreSQL.

16 ПРАКТИЧЕСКАЯ РАБОТА №16

16.1 Цель работы.

Изучение видов связей между сущностями в Hibernate. Использование транзакций.

16.2 Задание.

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

16.3 Код программы.

```
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.

```

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 + "\n" +
            ", address=" + address + "\n" +
            '};
    }
}

```

Листинг 16.2 – код класса Bank.

```

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.

16.4 Тестирование.

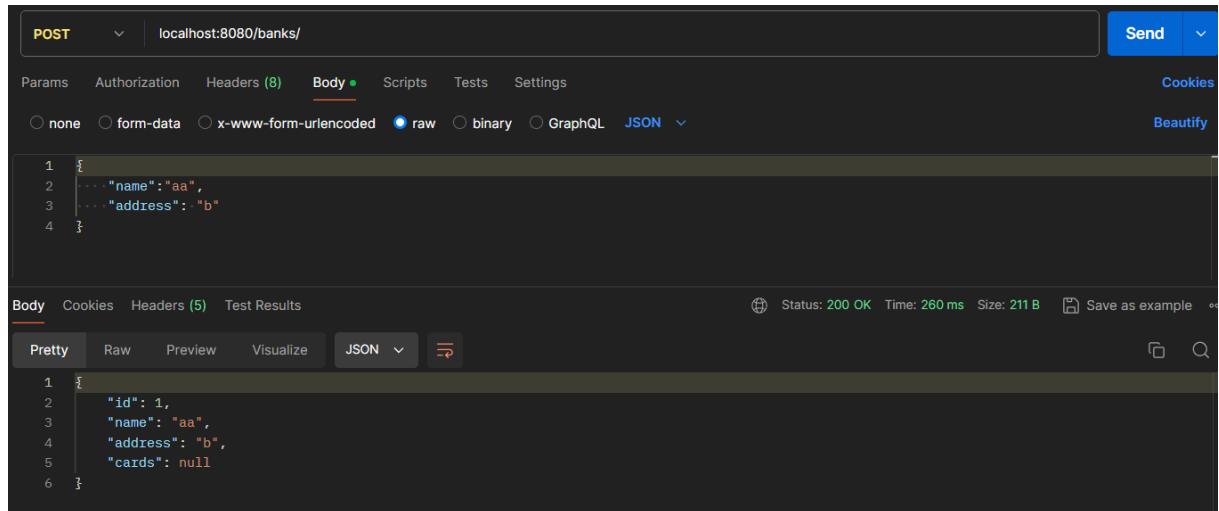


Рисунок 16.1 – результат работы программы.

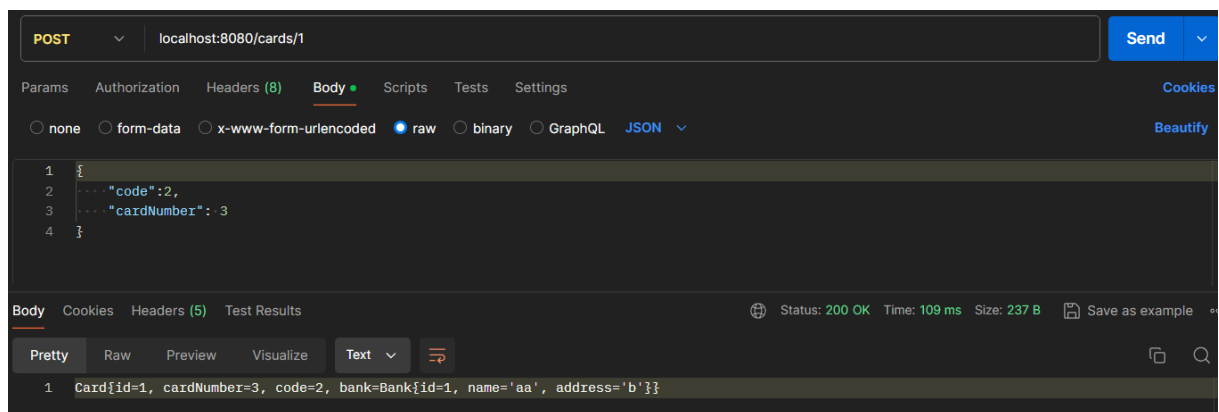


Рисунок 16.2 – результат работы программы.

16.5 Вывод.

Изучил связи между сущностями в Hibernate, использовал транзакции.

17 ПРАКТИЧЕСКАЯ РАБОТА №17

17.1 Цель работы.

Знакомство с Criteria API в Hibernate.

17.2 Задание.

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

17.3 Код программы.

```
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.

```
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.

17.4 Тестирование.

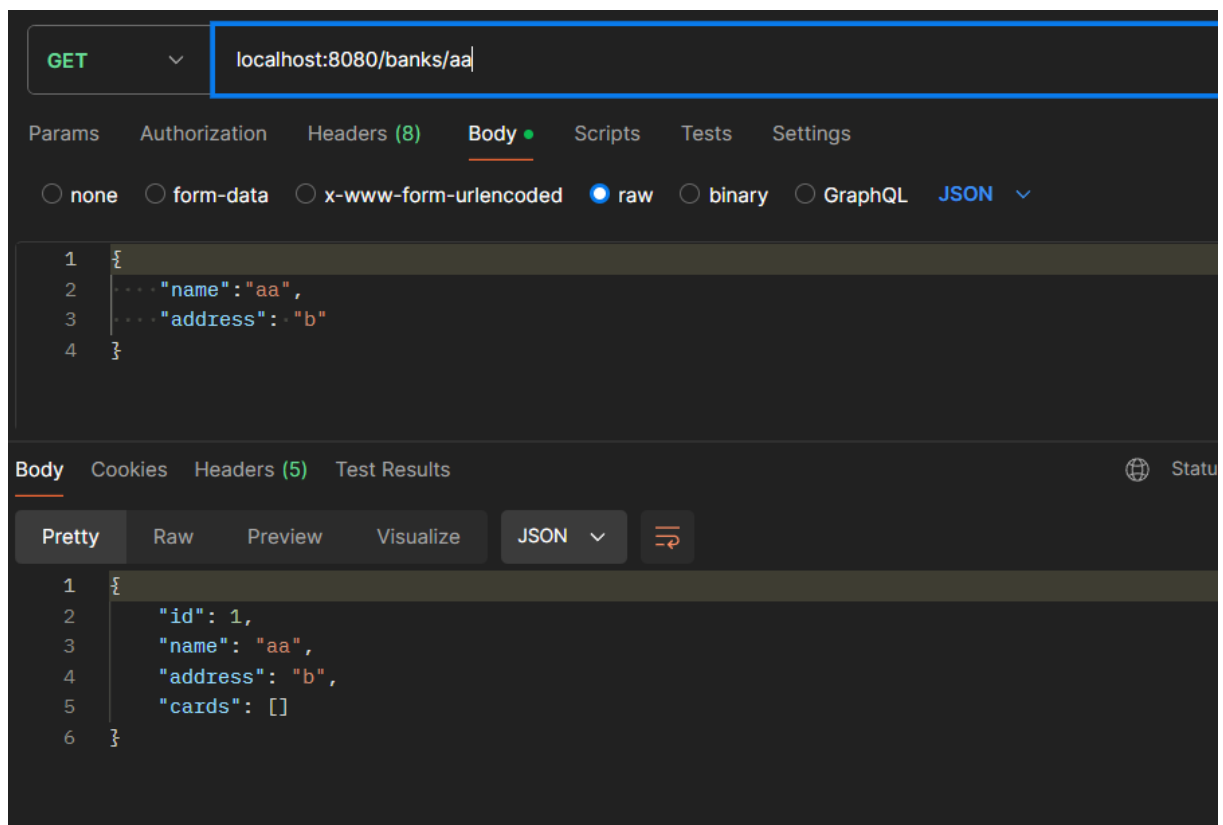


Рисунок 17 – результат работы программы.

17.5 Вывод.

Познакомился с Criteria API в Hibernate.

18 ПРАКТИЧЕСКАЯ РАБОТА №18

18.1 Цель работы.

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

18.2 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.

18.3 Код программы.

```
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.

```
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.

```

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;

    @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.

```

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.

18.4 Тестирование.

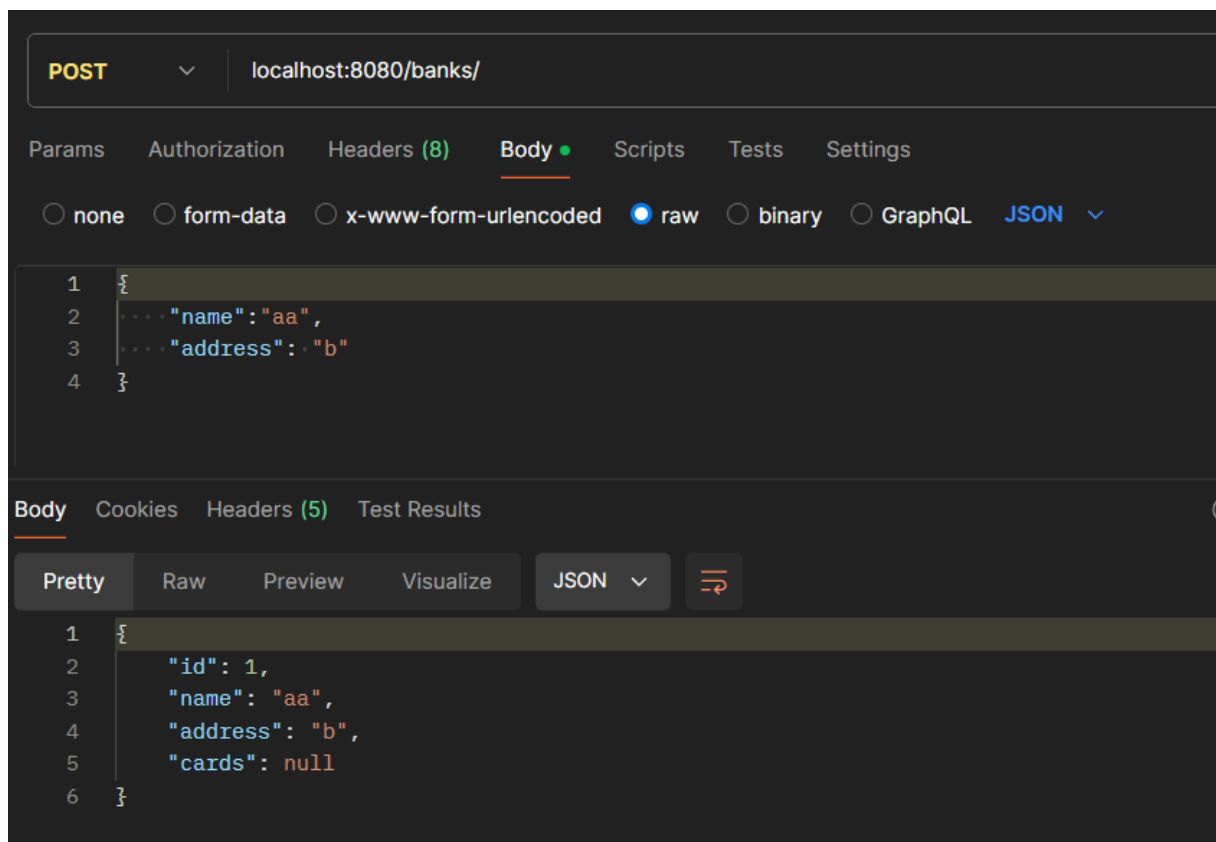


Рисунок 18 – результат работы программы.

18.5 Вывод.

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

19 ПРАКТИЧЕСКАЯ РАБОТА №19

19.1 Цель работы.

Знакомство с логированием с использованием Logback в Spring.

19.2 Задание.

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

19.3 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.

```
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();
    }

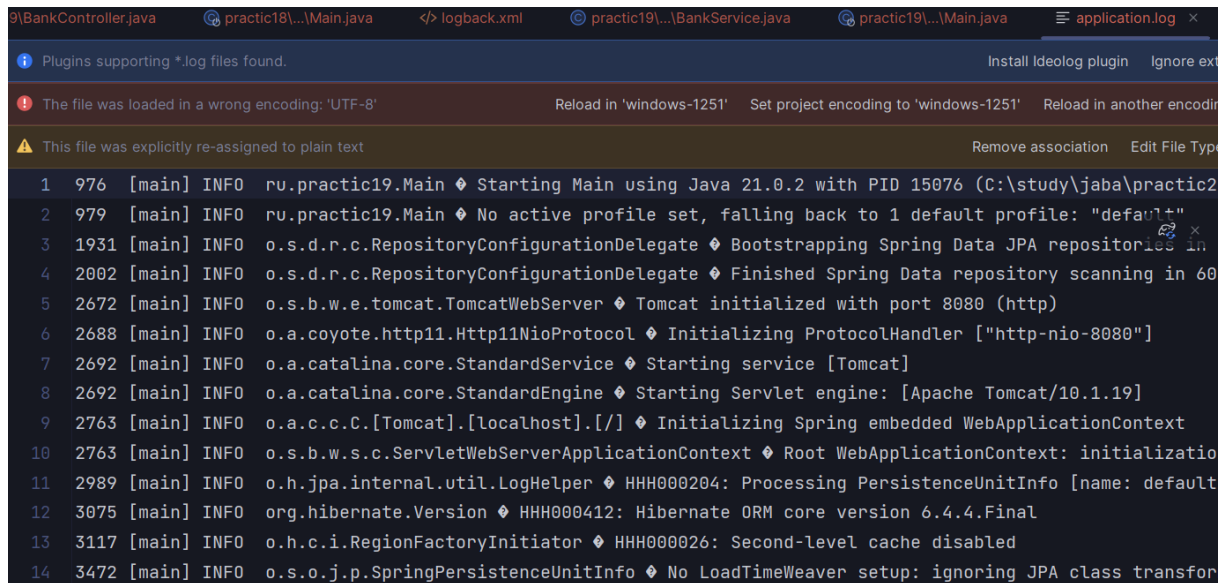
    public Bank saveBank(Bank bank){
        log.info("saving bank");
        return bankRepository.save(bank);
    }

    public void deleteById(Long id){
        log.info("deleting bank");
        bankRepository.deleteById(id);
    }
}

```

Листинг 19.2 – код класса BankService.

19.4 Тестирование.



The screenshot shows an IDE window with a log file named 'application.log'. The log contains the following entries:

```
1 976 [main] INFO ru.practic19.Main ♦ Starting Main using Java 21.0.2 with PID 15076 (C:\study\java\practic2
2 979 [main] INFO ru.practic19.Main ♦ No active profile set, falling back to 1 default profile: "default"
3 1931 [main] INFO o.s.d.r.c.RepositoryConfigurationDelegate ♦ Bootstrapping Spring Data JPA repositories in
4 2002 [main] INFO o.s.d.r.c.RepositoryConfigurationDelegate ♦ Finished Spring Data repository scanning in 60
5 2672 [main] INFO o.s.b.w.e.tomcat.TomcatWebServer ♦ Tomcat initialized with port 8080 (http)
6 2688 [main] INFO o.a.coyote.http11.Http11NioProtocol ♦ Initializing ProtocolHandler ["http-nio-8080"]
7 2692 [main] INFO o.a.catalina.core.StandardService ♦ Starting service [Tomcat]
8 2692 [main] INFO o.a.catalina.core.StandardEngine ♦ Starting Servlet engine: [Apache Tomcat/10.1.19]
9 2763 [main] INFO o.a.c.c.C.[Tomcat].[localhost].[/] ♦ Initializing Spring embedded WebApplicationContext
10 2763 [main] INFO o.s.b.w.s.c.ServletWebServerApplicationContext ♦ Root WebApplicationContext: initializatio
11 2989 [main] INFO o.h.jpa.internal.util.LogHelper ♦ HHH000204: Processing PersistenceUnitInfo [name: default
12 3075 [main] INFO org.hibernate.Version ♦ HHH000412: Hibernate ORM core version 6.4.4.Final
13 3117 [main] INFO o.h.c.i.RegionFactoryInitiator ♦ HHH000026: Second-level cache disabled
14 3472 [main] INFO o.s.o.j.p.SpringPersistenceUnitInfo ♦ No LoadTimeWeaver setup: ignoring JPA class transfor
```

Рисунок 19 – результат работы программы.

19.5 Вывод.

Познакомился с логированием Logback в Spring.

20 ПРАКТИЧЕСКАЯ РАБОТА №20

20.1 Цель работы.

Использование Spring AOP. Pointcut, JoinPoint, Advice.

20.2 Задание.

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

20.3 Код программы.

```
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.task20.BankService.*(..)) || execution(* ru.task20.CardService.*(..))")
    public void beforeMethodExecution() {
        startTime = System.currentTimeMillis();
    }

    @After("execution(* ru.task20.BankService.*(..)) || execution(* ru.task20.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.

20.4 Тестирование.

```
tp-nio-8080-exec-2] INFO r.p.practic20.BankService > saving bank  
t into bank (address,name) values (?,?)  
tp-nio-8080-exec-2] INFO r.p.MethodExecutionTimeAspect ♦ MethodBank ru.practic20.BankService.saveBank(Bank) was executed in 71 ms
```

Рисунок 20 – результат работы программы.

20.5 Вывод.

Использованил Spring AOP. Pointcut, JoinPoint, Advice.

21 ПРАКТИЧЕСКАЯ РАБОТА №21

21.1 Цель работы.

Проксирование. Аннотация Transactional. Аннотация Async.

21.2 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.

21.3 Код программы.

```
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("vladimir_namestnikov@vk.com");
        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("vladimir_namestnikov@vk.com");
        mailMessage.setSubject("Message from spring");
        mailMessage.setText(card.toString());
    }
}
```



```

    javaMailSender.send(mailMessage);
}
}

```

Листинг 21.1 – код интерфейса EmailService.

```

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.

```

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=task21
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.

21.4 Тестирование.

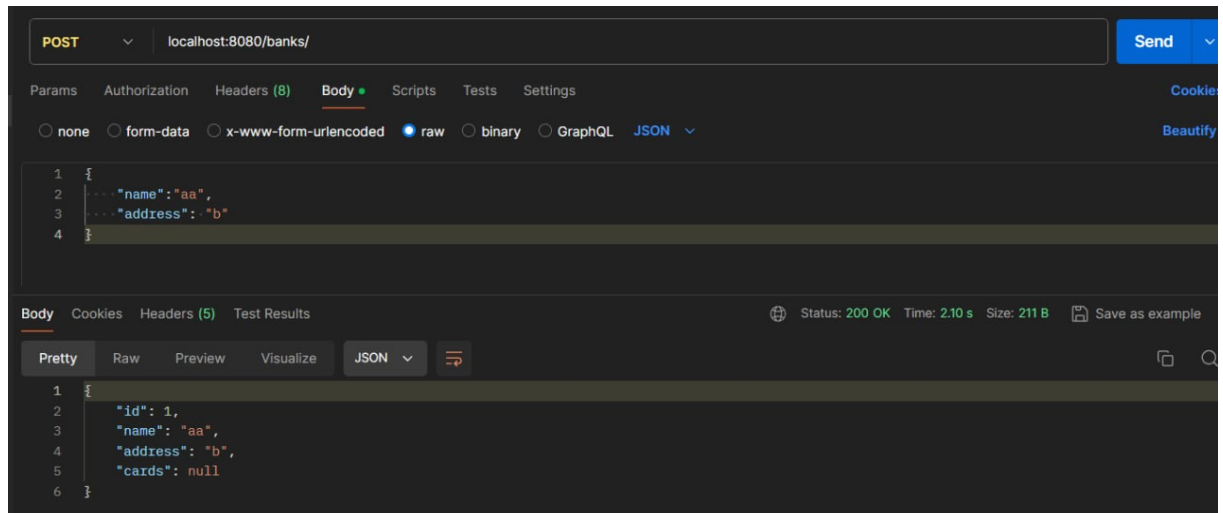


Рисунок 21.1 – результат работы программы.

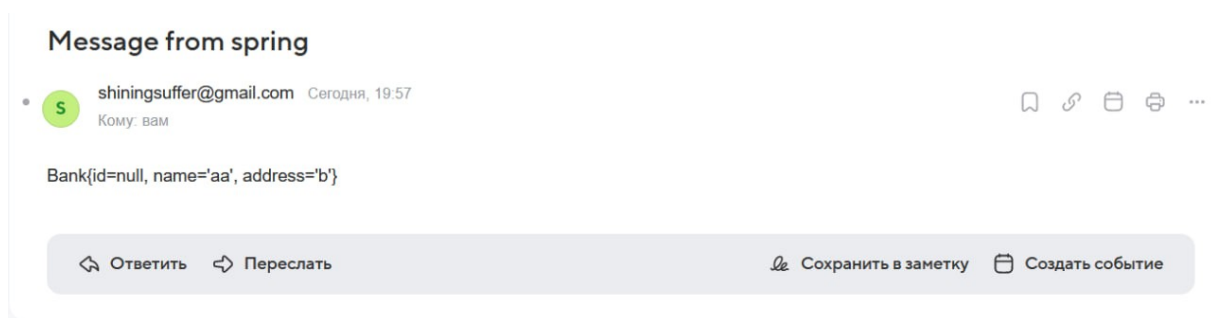


Рисунок 21.2 – результат работы программы.

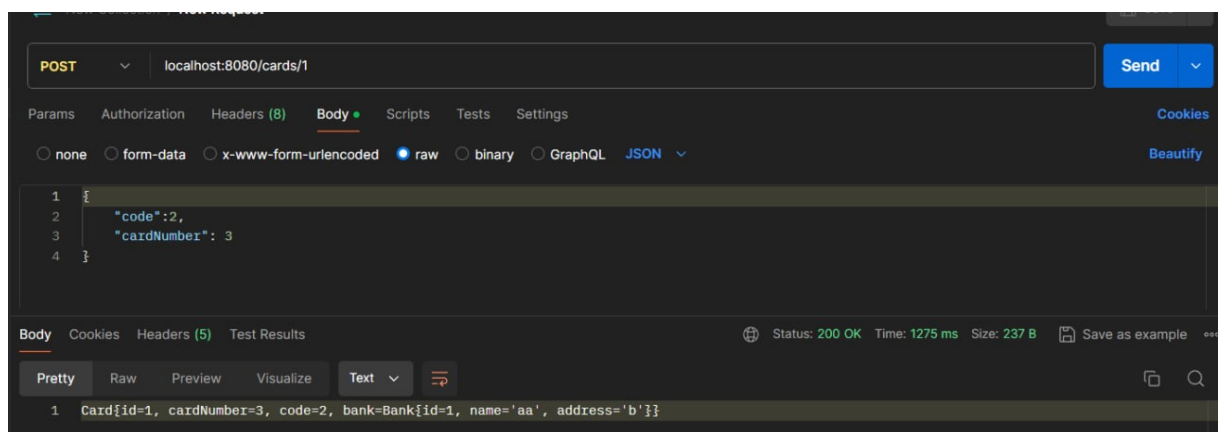



Рисунок 21.3 – результат работы программы.

Message from spring

-  **shiningsuffer@gmail.com** Сегодня, 19:58
Кому: вам

`Card{id=null, cardNumber=3, code=2, bank=null}`

Рисунок 21.4 – результат работы программы.

21.5 Вывод.

Проведена работа с проксированием, аннотацией `Transactional`, аннотацией `Async`.

22 ПРАКТИЧЕСКАЯ РАБОТА №22

22.1 Цель работы.

Планирование заданий. Scheduler в Spring.

22.2 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).

22.3 Код программы.

```
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.task22:type=ScheduledTask");
```

```

    if (!mbs.isRegistered(name)) {
        mbs.registerMBean(this, name);
    } else {
        log.info("MBean with name {} is already registered.", name);
    }
}

@Scheduled(fixedRateString = "PT1M")
public void remadeFiles() throws IOException {

Files.walk(Path.of
("C:/study/jaba/task2022/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/task22/src/main/resources/entities/banks.txt");
    banksFile.write(bankService.getBanks().toString());

    FileWriter cardsFile = new
FileWriter("C:/study/jaba/task22/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.

```

import java.io.IOException;

public interface TaskMXBean {

```

```
void runRemadeFiles() throws IOException;  
}
```

Листинг 22.2 – код интерфейса TaskMXBean.

22.4 Тестирование.

```
1 [{"Bank{id=1, name='aa', address='b'}"]
```

Рисунок 22 – результат работы программы.

22.5 Вывод.

Получены навыки планирования заданий в Spring через Scheduler.

23 ПРАКТИЧЕСКАЯ РАБОТА №23

23.1 Цель работы.

Использование Spring Security для аутентификации и авторизации пользователей.

23.2 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 неавторизованных пользователей.

23.3 Код программы.

```
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.

```

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class CustomUser {
    private String name;
    private String password;
}

```

Листинг 23.2 – код класса CustomUser.

```

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("!!!! ERRRRROOOOORRR !!!!");
        }
        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.

```

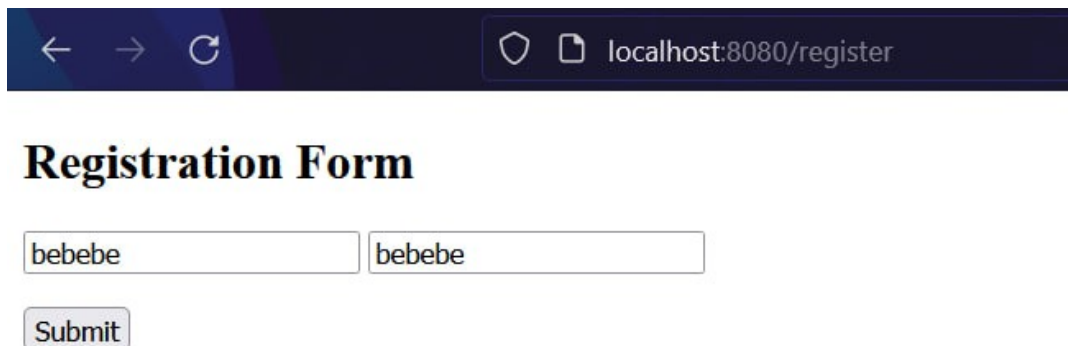
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Registration Form</title>
</head>
<body>
<h2>Registration Form</h2>
<form action="/register" th:action="@{/register}" th:object="${customUser}"
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.

23.4 Тестирование.



← → ↻ localhost:8080/register

Registration Form

Рисунок 23.1 – результат работы программы.

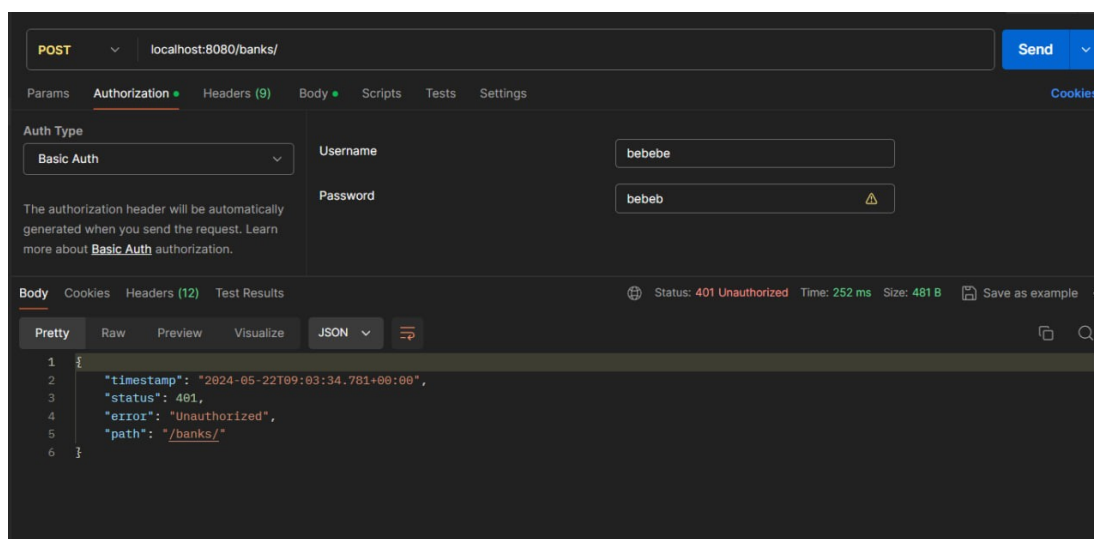


Рисунок 23.2 – результат работы программы.

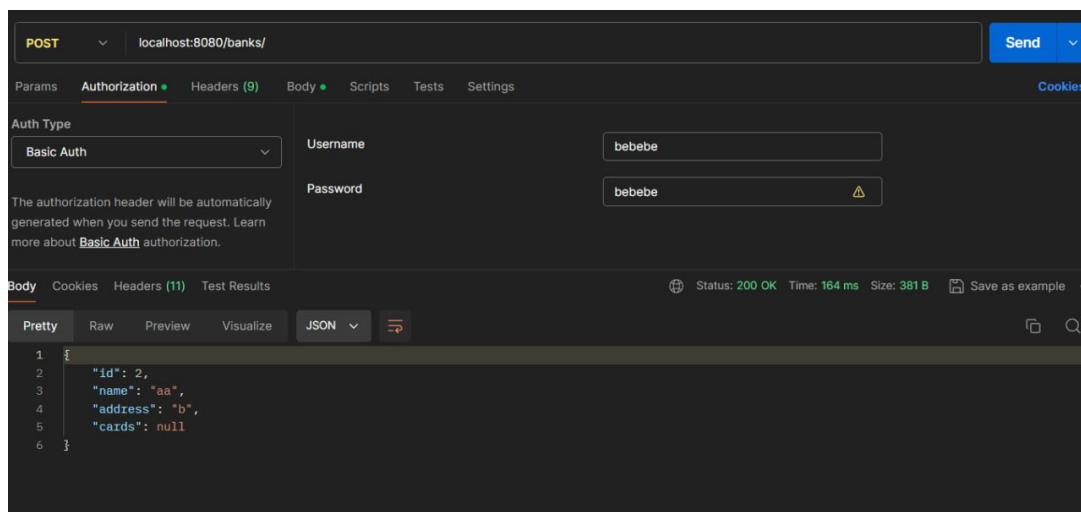


Рисунок 23.3 – результат работы программы.

23.5 Вывод.

Научился пользоваться Spring Security для аутентификации и авторизации пользователей.

24 ПРАКТИЧЕСКАЯ РАБОТА №24

24.1 Цель работы.

Тестирование в Spring Framework с использованием Junit.

24.2 Задание.

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

24.3 Код программы.

```
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.

```
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.

24.4 Тестирование.

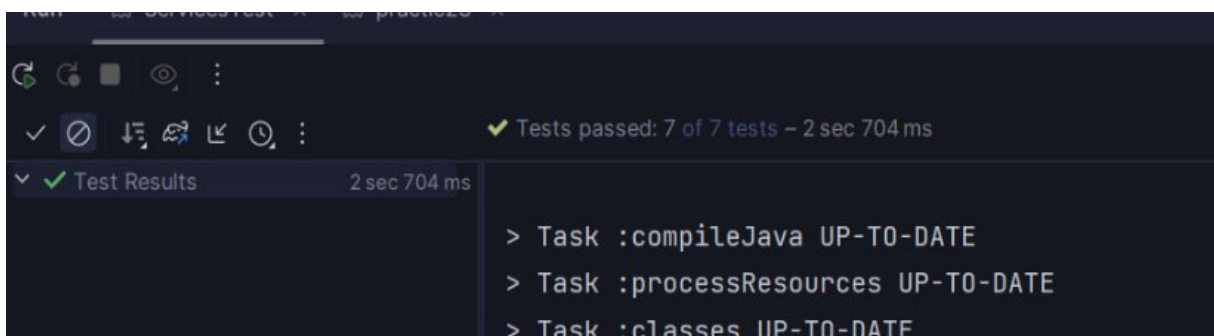


Рисунок 24 – результат работы программы.

24.5 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 "Вильямс", 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: 27.05.24). 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: 27.05.24). Заголовок с экрана.
4. Паттерны проектирования на Java. Каталог Java-примеров. [Электронный ресурс]. URL: <https://refactoring.guru/ru/design-patterns/java> (дата обращения: 27.05.24). Заголовок с экрана.
5. Руководство по Spring [Электронный ресурс]. URL: <https://proselyte.net/tutorials/spring-tutorial-full-version/> (дата обращения: 27.05.24). Заголовок с экрана.
6. The Reactive Manifesto [Электронный ресурс]. URL: <https://www.reactivemanifesto.org/> (дата обращения: 27.05.24). Заголовок с экрана.
7. Spring Framework Documentation [Электронный ресурс]. URL: <https://docs.spring.io/spring-framework/docs/current/reference/html/web.html> (дата обращения: 27.05.24). 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: 27.05.24). Заголовок с экрана.