



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

---

---

Институт информационных технологий (ИТ)

Кафедра инструментального и прикладного программного обеспечения (ИиППО)

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

**по дисциплине**

**«Шаблоны программных платформ языка Java»**

**Вариант 21**

Выполнил студент группы ИКБО-20-19

Николаев-Аксенов И. С.

Принял  
*Ассистент*

Батанов А. О.

Практические работы  
выполнены

«\_\_»\_\_\_\_\_2021 г.

(подпись студента)

«Зачтено»

«\_\_»\_\_\_\_\_2021 г.

(подпись руководителя)

Москва 2021

## Содержание

Практическая работа №1 .....	4
Практическая работа №2 .....	6
Практическая работа №3 .....	10
Практическая работа №4 .....	17
Практическая работа №5 .....	20
Практическая работа №6 .....	23
Практическая работа №7 .....	37
Практическая работа №8 .....	43
Практическая работа №9 .....	50
Практическая работа №10 .....	53
Практическая работа №11 .....	56
Практическая работа №12 .....	59
Практическая работа №13 .....	62
Практическая работа №14 .....	64
Практическая работа №15 .....	73
Практическая работа №16 .....	84
Практическая работа №17 .....	95
Практическая работа №18 .....	103
Практическая работа №19 .....	111
Практическая работа №20 .....	115
Практическая работа №21 .....	119
Практическая работа №22 .....	125
Практическая работа №23 .....	132

Практическая работа №24 .....	143
Вывод.....	146
Список использованных источников .....	146

## Практическая работа №1

### *Цель работы*

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

Постановка задачи: Имплементировать интерфейс Comparator, сравнивающий двух студентов по набранным за семестр баллов.

### *Листинг программы*

#### *Student.java*

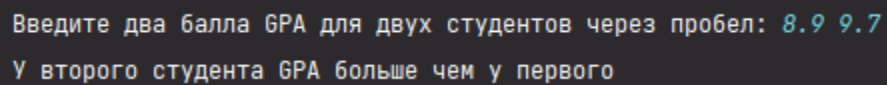
```
1. public class Student {
2.     private double gpa = 0;
3.
4.     public Student(double gpa) {
5.         this.gpa = gpa;
6.     }
7.     public double getGPA() {
8.         return gpa;
9.     }
10.    public void setGPA(double gpa) {
11.        this.gpa = gpa;
12.    }
13. }
14. 
```

#### *startStudent.java*

```
1. import java.util.Comparator;
2. import java.util.Scanner;
3.
4. public class startStudent {
5.     public static void main(String[] args) {
6.         System.out.print("Введите два балла GPA для двух студентов через пробел:
7.         ");
8.         Scanner sc = new Scanner(System.in);
9.         String[] arr = sc.nextLine().split(" ");
10.
11.         Student a = new Student(Double.parseDouble(arr[0]));
12.         Student b = new Student(Double.parseDouble(arr[1]));
13.
14.         Comparator<Student> compGPA = (o1, o2) -> Double.compare(o1.getGPA(),
15.             o2.getGPA());
16.         double result = compGPA.compare(a, b);
17.     }
18. }
```

```
15.
16.     if(result >= 1)
17.         System.out.println("У первого студента GPA больше чем у второго.");
18.     else if(result == 0)
19.         System.out.println("Оба студента имеют одинаковые баллы.");
20.     else
21.         System.out.println("У второго студента GPA больше чем у первого");
22.     }
23. }
24.
```

### ***Результат выполнения программы***

A screenshot of a program's output in a dark-themed terminal. The text is white and shows the program's instructions and the user's input. The instructions are "Введите два балла GPA для двух студентов через пробел:" and "У второго студента GPA больше чем у первого". The user's input is "8.9 9.7".

Введите два балла GPA для двух студентов через пробел: 8.9 9.7  
У второго студента GPA больше чем у первого

Рисунок 1.1 – Демонстрация работы программы

## Практическая работа №2

### *Цель работы*

Тема: Работа со Stream API в Java 8.

Постановка задачи: Уменьшение веса каждого объекта на 5, фильтрация по дате рождения меньшей, чем 3 февраля 1999, конкатенация фамилий в строку через пробел.

### *Листинг программы*

#### *Human.java*

```
1. import java.time.LocalDate;
2.
3. public class Human {
4.     private int age, weight;
5.     private String firstName, lastName;
6.     private LocalDate birthDate;
7.
8.     public Human(int age, String firstName, String lastName, LocalDate birthDate,
        int weight) {
9.         this.age = age;
10.        this.firstName = firstName;
11.        this.lastName = lastName;
12.        this.birthDate = birthDate;
13.        this.weight = weight;
14.    }
15.
16.    public int getAge() {
17.        return age;
18.    }
19.
20.    public void setAge(int age) {
21.        this.age = age;
22.    }
23.
24.    public int getWeight() {
25.        return weight;
26.    }
27.
28.    public void setWeight(int weight) {
29.        this.weight = weight;
30.    }
31.
32.    public String getFirstName() {
33.        return firstName;
34.    }
```

```

35.
36.     public void setFirstName(String firstName) {
37.         this.firstName = firstName;
38.     }
39.
40.     public String getLastName() {
41.         return lastName;
42.     }
43.
44.     public void setLastName(String lastName) {
45.         this.lastName = lastName;
46.     }
47.
48.     public LocalDate getBirthDate() {
49.         return birthDate;
50.     }
51.
52.     public void setBirthDate(LocalDate birthDate) {
53.         this.birthDate = birthDate;
54.     }
55.
56.     @Override
57.     public String toString() {
58.         return firstName + " " + lastName + ", " + age + " years, " + birthDate +
59.             ", " + weight + "kg";
60.     }

```

## *HumanBuilder.java*

```

1. import java.time.LocalDate;
2. import java.util.*;
3. import java.util.stream.Stream;
4.
5. public class HumanBuilder {
6.     private String[] firstNames = {"James", "Mary", "John", "Patricia", "Robert",
7.         "Jennifer", "Michael", "Linda", "William", "Elizabeth", "David", "Barbara",
8.         "Richard", "Susan", "Joseph", "Jessica", "Thomas", "Sarah", "Charles", "Karen",
9.         "Christopher", "Nancy", "Daniel", "Lisa", "Matthew", "Margaret", "Anthony",
10.        "Betty", "Donald", "Sandra", "Mark", "Ashley", "Paul", "Dorothy", "Steven",
11.        "Kimberly", "Andrew", "Emily", "Kenneth", "Donna", "Joshua", "Michelle", "Kevin",
12.        "Carol", "Brian", "Amanda"};
13.
14.     private String[] lastNames = {"Smith", "Johnson", "Williams", "Brown", "Jones",
15.        "Garcia", "Miller", "Davis", "Rodriguez", "Martinez", "Hernandez", "Lopez",
16.        "Gonzalez", "Wilson", "Anderson", "Thomas", "Taylor", "Moore", "Jackson", "Martin",
17.        "Lee", "Perez", "Thompson", "White", "Harris", "Sanchez", "Clark", "Ramirez",
18.        "Lewis", "Robinson"};
19.
20.     private List<Human> generateList(int size) {
21.         List<Human> result = new ArrayList<>();

```

```

11.         Random rand = new Random();
12.
13.         for(int i = 0; i < size; i++) {
14.             LocalDate ld = LocalDate.of(rand.nextInt(70) + 1950, rand.nextInt(12) +
1, rand.nextInt(27) + 1);
15.             result.add(new Human(rand.nextInt(99),
firstNames[rand.nextInt(firstNames.length)],
lastNames[rand.nextInt(lastNames.length)], ld, rand.nextInt(100) + 20));
16.         }
17.
18.         return result;
19.     }
20.
21.     public void humanTask(int size) {
22.         List<Human> harr = generateList(size);
23.
24.         System.out.println("Original list:");
25.         harr.forEach(System.out::println);
26.
27.         System.out.println("\nList after applying stream methods:");
28.         Stream<Human> s1 = harr.stream();
29.         Stream<Human> s2 = harr.stream();
30.         Stream<Human> s3 = harr.stream();
31.
32.         System.out.println("\nFirst stream: ");
33.         s1.peek(o -> o.setWeight(o.getWeight() - 5)).forEach(System.out::println);
34.
35.         System.out.println("\nSecond stream: ");
36.         s2.filter(o -> o.getBirthDate().isBefore(LocalDate.of(1999, 2,
3))) .forEach(System.out::println);
37.
38.         System.out.println("\nThird stream: ");
39.         s3.map(o -> o.getLastName() + ' ').forEach(System.out::print);
40.     }
41. }
42.

```

## *Main.java*

```

1. public class main {
2.     public static void main(String[] args) {
3.         HumanBuilder hb = new HumanBuilder();
4.         hb.humanTask(10);
5.     }
6. }
7.

```



## *Результат выполнения программы*

```
Original list:
Susan Clark, 94 years, 2004-03-22, 91kg
John Gonzalez, 90 years, 1990-07-21, 85kg
Donald Perez, 9 years, 1975-10-03, 61kg
Steven Taylor, 78 years, 1970-06-11, 51kg
Joseph Williams, 80 years, 2011-03-09, 89kg

List after applying stream methods:

First stream:
Susan Clark, 94 years, 2004-03-22, 86kg
John Gonzalez, 90 years, 1990-07-21, 80kg
Donald Perez, 9 years, 1975-10-03, 56kg
Steven Taylor, 78 years, 1970-06-11, 46kg
Joseph Williams, 80 years, 2011-03-09, 84kg

Second stream:
John Gonzalez, 90 years, 1990-07-21, 80kg
Donald Perez, 9 years, 1975-10-03, 56kg
Steven Taylor, 78 years, 1970-06-11, 46kg

Third stream:
Clark Gonzalez Perez Taylor Williams
Process finished with exit code 0
```

Рисунок 2.1 – Демонстрация работы программы

## Практическая работа №3

### *Цель работы*

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

Постановка задачи: Создать свои потокобезопасные имплементации интерфейсов в соответствии с вариантом индивидуального задания. Set с использованием Lock, Map с использованием Semaphore.

### *Листинг программы*

#### *MapSemaphore.java*

```
1. import java.util.Collection;
2. import java.util.HashMap;
3. import java.util.Map;
4. import java.util.Set;
5. import java.util.concurrent.Semaphore;
6.
7. public class MapSemaphore<K, V> extends HashMap<K, V> implements Map<K, V> {
8.     private static final Semaphore sem = new Semaphore(1);
9.
10.    @Override
11.    public int size() {
12.        return super.size();
13.    }
14.
15.    @Override
16.    public boolean isEmpty() {
17.        return super.isEmpty();
18.    }
19.
20.    @Override
21.    public boolean containsKey(Object key) {
22.        boolean b = false;
23.        try {
24.            sem.acquire();
25.            b = super.containsKey(key);
26.            sem.release();
27.        } catch (InterruptedException e) {
28.            e.printStackTrace();
29.        }
30.        return b;
31.    }
32.
```

```

33.     @Override
34.     public boolean containsValue(Object value) {
35.         boolean b = false;
36.         try {
37.             sem.acquire();
38.             b = super.containsValue(value);
39.             sem.release();
40.         } catch (InterruptedException e) {
41.             e.printStackTrace();
42.         }
43.         return b;
44.     }
45.
46.     @Override
47.     public V get(Object key) {
48.         V k = null;
49.         try {
50.             sem.acquire();
51.             k = super.get(key);
52.             sem.release();
53.         } catch (InterruptedException e) {
54.             e.printStackTrace();
55.         }
56.         return k;
57.     }
58.
59.     @Override
60.     public V put(K key, V value) {
61.         V k = null;
62.         try {
63.             sem.acquire();
64.             k = super.put(key, value);
65.             sem.release();
66.         } catch (InterruptedException e) {
67.             e.printStackTrace();
68.         }
69.         return k;
70.     }
71.
72.     @Override
73.     public V remove(Object key) {
74.         V k = null;
75.         try {
76.             sem.acquire();
77.             k = super.remove(key);
78.             sem.release();
79.         } catch (InterruptedException e) {
80.             e.printStackTrace();
81.         }
82.         return k;

```

```

83.     }
84.
85.     @Override
86.     public void putAll(Map<? extends K, ? extends V> m) {
87.         try {
88.             sem.acquire();
89.             super.putAll(m);
90.             sem.release();
91.         } catch (InterruptedException e) {
92.             e.printStackTrace();
93.         }
94.     }
95.
96.     @Override
97.     public void clear() {
98.         try {
99.             sem.acquire();
100.            super.clear();
101.            sem.release();
102.        } catch (InterruptedException e) {
103.            e.printStackTrace();
104.        }
105.    }
106.
107.    @Override
108.    public Set<K> keySet() {
109.        Set<K> k = null;
110.        try {
111.            sem.acquire();
112.            k = super.keySet();
113.            sem.release();
114.        } catch (InterruptedException e) {
115.            e.printStackTrace();
116.        }
117.        return k;
118.    }
119.
120.    @Override
121.    public Collection<V> values() {
122.        Collection<V> k = null;
123.        try {
124.            sem.acquire();
125.            k = super.values();
126.            sem.release();
127.        } catch (InterruptedException e) {
128.            e.printStackTrace();
129.        }
130.        return k;
131.    }
132.

```

```

133.         @Override
134.         public Set<Entry<K, V>> entrySet() {
135.             Set<Entry<K, V>> k = null;
136.             try {
137.                 sem.acquire();
138.                 k = super.entrySet();
139.                 sem.release();
140.             } catch (InterruptedException e) {
141.                 e.printStackTrace();
142.             }
143.             assert k != null;
144.             return k;
145.         }
146.     }
147.

```

## *SetLock.java*

```

1. import java.util.ArrayList;
2. import java.util.Collection;
3. import java.util.Iterator;
4. import java.util.Set;
5. import java.util.concurrent.locks.ReentrantLock;
6.
7. public class SetLock<E> implements Set {
8.     private ArrayList<E> arr = new ArrayList<>();
9.     private static final ReentrantLock re = new ReentrantLock();
10.
11.     @Override
12.     public boolean add(Object o) {
13.         re.lock();
14.         boolean b = arr.add((E) o);
15.         re.unlock();
16.         return b;
17.     }
18.
19.     @Override
20.     public boolean remove(Object o) {
21.         re.lock();
22.         boolean b = arr.remove(o);
23.         re.unlock();
24.         return b;
25.     }
26.
27.     @Override
28.     public boolean addAll(Collection c) {
29.         re.lock();
30.         boolean b = arr.addAll(c);
31.         re.unlock();
32.         return b;

```

```

33.     }
34.
35.     @Override
36.     public void clear() {
37.         re.lock();
38.         arr.clear();
39.         re.unlock();
40.     }
41.
42.     @Override
43.     public boolean removeAll(Collection c) {
44.         re.lock();
45.         boolean b = arr.removeAll(c);
46.         re.unlock();
47.         return b;
48.     }
49.
50.     @Override
51.     public boolean retainAll(Collection c) {
52.         re.lock();
53.         boolean b = arr.retainAll(c);
54.         re.unlock();
55.         return b;
56.     }
57.
58.     @Override
59.     public boolean containsAll(Collection c) {
60.         re.lock();
61.         boolean b = arr.containsAll(c);
62.         re.unlock();
63.         return b;
64.     }
65.
66.     @Override
67.     public boolean contains(Object o) {
68.         re.lock();
69.         boolean b = arr.contains((E) o);
70.         re.unlock();
71.         return b;
72.     }
73.
74.     @Override
75.     public int size() {
76.         return arr.size();
77.     }
78.
79.     @Override
80.     public boolean isEmpty() {
81.         return arr.isEmpty();
82.     }

```

```

83.
84.     @Override
85.     public Iterator iterator() {
86.         return arr.iterator();
87.     }
88.
89.     @Override
90.     public Object[] toArray() {
91.         return arr.toArray();
92.     }
93.
94.     @Override
95.     public Object[] toArray(Object[] a) {
96.         return arr.toArray(a);
97.     }
98.
99.     @Override
100.    public String toString() {
101.        return "SetLock{" +
102.            "arr=" + arr +
103.            '}';
104.    }
105. }
106.

```

## *TypesTester.java*

```

1. public class TypesTester {
2.     static class t1 extends Thread {
3.         @Override
4.         public void run() {
5.             SetLock<Integer> sl = new SetLock<>();
6.
7.             sl.add(5);
8.             sl.add(15);
9.             sl.add(25);
10.
11.             System.out.println(sl);
12.             System.out.println(sl.contains(25));
13.             sl.remove(15);
14.             System.out.println(sl.size());
15.         }
16.     }
17.
18.     static class t2 extends Thread {
19.         @Override
20.         public void run() {
21.             MapSemaphore<Integer, String> ms = new MapSemaphore<>();
22.
23.             ms.put(1, "Ivan");

```

```

24.         ms.put(2, "Petr");
25.         ms.put(3, "Mikhail");
26.
27.         System.out.println(ms);
28.         System.out.println(ms.values());
29.         System.out.println(ms.keySet());
30.
31.         ms.remove(2);
32.
33.         System.out.println(ms.size());
34.     }
35. }
36. public static void main(String[] args) {
37.     t1 th11 = new t1();
38.     t1 th12 = new t1();
39.     th11.start();
40.     th12.start();
41.
42.     try {
43.         Thread.sleep(900);
44.     } catch (InterruptedException e) {
45.         e.printStackTrace();
46.     }
47.
48.     System.out.println("-----");
49.
50.     t2 th21 = new t2();
51.     t2 th22 = new t2();
52.     th21.start();
53.     th22.start();
54. }
55. }
56.

```

### *Результат выполнения программы*

```

SetLock{arr=[5, 15, 25]}
true
2
SetLock{arr=[5, 15, 25]}
true
2
-----
{1=Ivan, 2=Petr, 3=Mikhail}
{1=Ivan, 2=Petr, 3=Mikhail}
[Ivan, Petr, Mikhail]
[Ivan, Petr, Mikhail]
[1, 2, 3]
[1, 2, 3]
2
2

```

Рисунок 3.1 – Демонстрация работы программы



## Практическая работа №4

### Цель работы

Тема: Работа с ExecutorService, CompletableFuture.

Постановка задачи: Реализовать собственную имплементацию ExecutorService с единственным параметром конструктора — количеством потоков.

### Листинг программы

#### *RandomWordThread.java*

```
1. import java.util.Random;
2.
3. public class RandomWordThread implements Runnable {
4.     private String[] words = {"capricious", "roasted", "check", "box", "day",
        "debonair", "coordinated", "observe", "beds", "jail", "wide-eyed", "anger",
        "attraction", "slimy", "thoughtless", "time", "drab", "pushy", "smiling",
        "helpful", "understood", "truck", "hobbies", "delight", "launch", "acoustic",
        "troubled", "thick", "cattle", "explode", "large", "melt", "release", "bashful",
        "hurried", "animal", "bite-sized", "kneel", "unaccountable", "squealing", "show",
        "drown", "telling", "aunt", "low", "superficial", "wave", "breath", "succeed",
        "complain"};
5.     private Random random = new Random();
6.
7.     @Override
8.     public void run() {
9.         try {
10.             System.out.println(words[random.nextInt(words.length)]);
11.             Thread.sleep(random.nextInt(5001) + 500);
12.         } catch (InterruptedException e) {
13.             e.printStackTrace();
14.         }
15.     }
16. }
17.
```

## *ExecutorServiceHandler.java*

```
1. import java.util.Random;
2. import java.util.concurrent.ExecutorService;
3. import java.util.concurrent.Executors;
4.
5. public class ExecutorServiceHandler {
6.     private ExecutorService exec;
7.     private Random random = new Random();
8.
9.     public ExecutorServiceHandler(int number) {
10.         exec = Executors.newFixedThreadPool(number);
11.
12.         int bound = random.nextInt(30) + 1;
13.         System.out.println("Запуск " + bound + " потоков, при возможных " + number
+ " потоках..\n");
14.
15.         for(int i = 0; i < bound; i++) {
16.             System.out.println("Запуск потока №" + (i + 1));
17.             exec.execute(new RandomWordThread());
18.         }
19.         exec.shutdown();
20.         System.out.println("Завершение работы потоков...");
21.     }
22. }
23.
```

## *Main.java*

```
1. public class Main {
2.     public static void main(String[] args) {
3.         ExecutorServiceHandler esh = new ExecutorServiceHandler(3);
4.     }
5. }
6.
```

## *Результат выполнения программы*

```
Запуск 12 потоков, при возможных 3 потоках..  
  
Запуск потока №1  
Запуск потока №2  
Запуск потока №3  
delight  
Запуск потока №4  
helpful  
Запуск потока №5  
Запуск потока №6  
Запуск потока №7  
Запуск потока №8  
Запуск потока №9  
Запуск потока №10  
Запуск потока №11  
animal  
Запуск потока №12  
Завершение работы потоков...  
animal  
cattle  
succeed  
explode  
acoustic  
complain  
pushy  
bashful  
slimy
```

Рисунок 4.1 – Демонстрация работы программы

## Практическая работа №5

### *Цель работы*

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

Постановка задачи: Реализовать паттерн Singleton как минимум 3-мя способами.

### *Листинг программы*

#### *FirstSingletonImpl.java*

```
1. public class FirstSingletonImpl {
2.     private static FirstSingletonImpl instance;
3.
4.     private FirstSingletonImpl() {
5.         System.out.println("FirstSingletonImpl");
6.     }
7.
8.     public static FirstSingletonImpl getInstance() {
9.         if(instance == null)
10.            instance = new FirstSingletonImpl();
11.         return instance;
12.     }
13. }
14.
```

#### *SecondSingletonImpl.java*

```
1. public class SecondSingletonImpl {
2.     private static SecondSingletonImpl instance = new SecondSingletonImpl();
3.
4.     private SecondSingletonImpl() {
5.         System.out.println("SecondSingletonImpl");
6.     }
7.
8.     public static SecondSingletonImpl getInstance() {
9.         return instance;
10.    }
11. }
12.
```

### ***ThirdSingletonImpl.java***

```
1. public class ThirdSingletonImpl {
2.     private ThirdSingletonImpl() {
3.         System.out.println("ThirdSingletonImpl");
4.     }
5.
6.     private static class ThirdSingletonImplHolder {
7.         private final static ThirdSingletonImpl instance = new
ThirdSingletonImpl();
8.     }
9.
10.    public static ThirdSingletonImpl getInstance() {
11.        return ThirdSingletonImplHolder.instance;
12.    }
13. }
14.
```

### ***FourthSingletonImpl.java***

```
1. public class FourthSingletonImpl {
2.     private static volatile FourthSingletonImpl instance;
3.
4.     private FourthSingletonImpl() {
5.         System.out.println("FourthSingletonImpl");
6.     }
7.
8.     public static FourthSingletonImpl getInstance() {
9.         if (instance == null)
10.            synchronized (FourthSingletonImpl.class) {
11.                if (instance == null)
12.                    instance = new FourthSingletonImpl();
13.            }
14.        return instance;
15.    }
16. }
17.
```

### ***Main.java***

```
1. public class Main {
2.     public static void main(String[] args) {
3.         FirstSingletonImpl.getInstance();
4.         SecondSingletonImpl.getInstance();
5.         ThirdSingletonImpl.getInstance();
6.         FourthSingletonImpl.getInstance();
7.     }
8. }
9.
```

### *Результат выполнения программы*

```
FirstSingletonImpl  
SecondSingletonImpl  
ThirdSingletonImpl  
FourthSingletonImpl
```

Рисунок 5.1 – Демонстрация работы программы

## Практическая работа №6

### *Цель работы*

Тема: Знакомство с реализацией порождающих паттернов проектирования.

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

### *Листинг программы*

#### *AbstractFactory:*

##### *AfricanGarden.java*

```
1. package AbstractFactory;
2.
3. public class AfricanGarden implements GardenFactory {
4.     public AfricanGarden() {
5.         System.out.println("Создан африканский сад!");
6.     }
7.
8.     @Override
9.     public Tree plantTree() {
10.         return new Baobab();
11.     }
12.
13.     @Override
14.     public Flower plantFlower() {
15.         return new Gloriosa();
16.     }
17. }
18.
```

##### *Baobab.java*

```
1. package AbstractFactory;
2.
3. public class Baobab implements Tree {
4.     public Baobab() {
5.         System.out.println("Посажено Дерево - Баобаb");
6.     }
7. }
8.
```

## *EuropeanGarden.java*

```
1. package AbstractFactory;
2.
3. public class EuropeanGarden implements GardenFactory {
4.     public EuropeanGarden() {
5.         System.out.println("Создан европейский сад!");
6.     }
7.
8.     @Override
9.     public Tree plantTree() {
10.        return new Oak();
11.    }
12.
13.    @Override
14.    public Flower plantFlower() {
15.        return new Rose();
16.    }
17. }
18.
```

## *Flower.java*

```
1. package AbstractFactory;
2.
3. public interface Flower {
4. }
5.
```

## *GardenFactory.java*

```
1. package AbstractFactory;
2.
3. public interface GardenFactory {
4.     Tree plantTree();
5.     Flower plantFlower();
6. }
7.
```



## *Gloriosa.java*

```
1. package AbstractFactory;
2.
3. public class Gloriosa implements Flower {
4.     public Gloriosa() {
5.         System.out.println("Посажен Цветок - Глориоза");
6.     }
7. }
8.
```

## *HomeGarden.java*

```
1. package AbstractFactory;
2.
3. public class HomeGarden implements GardenFactory {
4.     public HomeGarden() {
5.         System.out.println("Создан домашний сад");
6.     }
7.
8.     @Override
9.     public Tree plantTree() {
10.        System.out.println("Вы не можете посадить дома дерево!");
11.        return null;
12.    }
13.
14.    @Override
15.    public Flower plantFlower() {
16.        return new Orchid();
17.    }
18. }
19.
```

## *MainAbstractFactory.java*

```
1. package AbstractFactory;
2.
3. public class MainAbstractFactory {
4.     public static void main(String[] args) {
5.         AfricanGarden ag = new AfricanGarden();
6.         ag.plantFlower();
7.         ag.plantTree();
8.
9.         System.out.print("\n");
10.        EuropeanGarden eg = new EuropeanGarden();
11.        eg.plantFlower();
12.        eg.plantTree();
13.
14.        System.out.print("\n");
15.        HomeGarden hg = new HomeGarden();
```

```
16.         hg.plantFlower();
17.         hg.plantTree();
18.     }
19. }
20.
```

### ***Oak.java***

```
1. package AbstractFactory;
2.
3. public class Oak implements Tree {
4.     public Oak() {
5.         System.out.println("Посажено Дерево - Дуб");
6.     }
7. }
8.
```

### ***Orchid.java***

```
1. package AbstractFactory;
2.
3. public class Orchid implements Flower {
4.     public Orchid() {
5.         System.out.println("Посажен Цветок - Орхидея");
6.     }
7. }
8.
```

### ***Rose.java***

```
1. package AbstractFactory;
2.
3. public class Rose implements Flower {
4.     public Rose() {
5.         System.out.println("Посажен Цветок - Роза");
6.     }
7. }
8.
```

### ***Tree.java***

```
1. package AbstractFactory;
2.
3. public interface Tree {
4. }
5.
```

## ***Builder:***

### ***Animal.java***

```
1. package Builder;
2.
3. public class Animal {
4.     private final String species;
5.     private final int weight;
6.     private final String habitat;
7.     private final int lifespan;
8.
9.     public static class AnimalBuilder {
10.         private final String species;
11.         private int weight = 0;
12.         private String habitat = "";
13.         private int lifespan = 0;
14.
15.         public AnimalBuilder(String species) {
16.             this.species = species;
17.         }
18.
19.         public AnimalBuilder weight(int weight) {
20.             this.weight = weight;
21.             return this;
22.         }
23.
24.         public AnimalBuilder habitat(String habitat) {
25.             this.habitat = habitat;
26.             return this;
27.         }
28.
29.         public AnimalBuilder lifespan(int lifespan) {
30.             this.lifespan = lifespan;
31.             return this;
32.         }
33.
34.         public Animal create() {
35.             return new Animal(this);
36.         }
37.     }
38.
39.     private Animal(AnimalBuilder builder) {
40.         species = builder.species;
41.         weight = builder.weight;
42.         habitat = builder.habitat;
43.         lifespan = builder.lifespan;
44.     }
45.
46.     @Override
```

```

47.     public String toString() {
48.         return species + " обычно живет в " + habitat
49.             + ", весит примерно " + weight
50.             + " и живет " + lifespan + " лет.";
51.     }
52. }
53.

```

## *MainBuilder.java*

```

1. package Builder;
2.
3. public class MainBuilder {
4.     public static void main(String[] args) {
5.         Animal lion = new Animal
6.             .AnimalBuilder("Лев")
7.             .weight(150)
8.             .habitat("Саванна")
9.             .lifespan(12)
10.            .create();
11.
12.         Animal polarBear = new Animal
13.             .AnimalBuilder("Белый медведь")
14.             .weight(400)
15.             .habitat("Арктика")
16.             .lifespan(25)
17.             .create();
18.
19.         Animal redFox = new Animal
20.             .AnimalBuilder("Обыкновенная лисица")
21.             .weight(10)
22.             .habitat("тундра, и степи, и леса разного типа, и пустыни и
23.                 высокогорья")
24.             .lifespan(4)
25.             .create();
26.         System.out.println(lion);
27.         System.out.println(polarBear);
28.         System.out.println(redFox);
29.     }
30. }
31.

```

## ***FactoryMethod:***

### ***AfricanCow.java***

```
1. package FactoryMethod;
2.
3. public class AfricanCow extends Cow {
4.     @Override
5.     public String getType() {
6.         return "Cow from Africa";
7.     }
8. }
9.
```

### ***AfricanLion.java***

```
1. package FactoryMethod;
2.
3. public class AfricanLion extends Lion {
4.     @Override
5.     public String getType() {
6.         return "Lion from Africa";
7.     }
8. }
9.
```

### ***AfricanOwl.java***

```
1. package FactoryMethod;
2.
3. public class AfricanOwl extends Owl {
4.     @Override
5.     public String getType() {
6.         return "Owl from Africa";
7.     }
8. }
```

### ***AfricanZebra.java***

```
1. package FactoryMethod;
2.
3. public class AfricanZebra extends Zebra {
4.     @Override
5.     public String getType() {
6.         return "Zebra from Africa";
7.     }
8. }
9.
```

## *AfricanZoo.java*

```
1. package FactoryMethod;
2.
3. public class AfricanZoo extends Zoo {
4.     @Override
5.     protected Animal createAnimal(AnimalType type) {
6.         Animal animal = null;
7.
8.         switch (type) {
9.             case ZEBRA:
10.                animal = new AfricanZebra();
11.                break;
12.             case LION:
13.                animal = new AfricanLion();
14.                break;
15.             case COW:
16.                animal = new AfricanCow();
17.                break;
18.             case OWL:
19.                animal = new AfricanOwl();
20.                break;
21.             case BEAR:
22.                animal = new RussianBear();
23.                break;
24.         }
25.         return animal;
26.     }
27. }
28.
```

## *Animal.java*

```
1. package FactoryMethod;
2. public interface Animal {
3.     String getType();
4. }
5.
```

## *AnimalType.java*

```
1. package FactoryMethod;
2.
3. public enum AnimalType {
4.     ZEBRA,
5.     LION,
6.     COW,
7.     OWL,
8.     BEAR
9. }
```

## ***Bear.java***

```
1. package FactoryMethod;
2.
3. public class Bear implements Animal {
4.     @Override
5.     public String getType() {
6.         return "Bear";
7.     }
8. }
9.
```

## ***Cow.java***

```
1. package FactoryMethod;
2.
3. public class Cow implements Animal {
4.     @Override
5.     public String getType() {
6.         return "Cow";
7.     }
8. }
```

## ***Lion.java***

```
1. package FactoryMethod;
2.
3. public class Lion implements Animal {
4.     @Override
5.     public String getType() {
6.         return "Lion";
7.     }
8. }
9.
```

## ***MainFactoryMethod.java***

```
1. package FactoryMethod;
2.
3. public class MainFactoryMethod {
4.     public static void main(String[] args) {
5.         System.out.println("Zoo in Cape Town, Republic of South Africa");
6.         Zoo capeTown = new AfricanZoo();
7.         capeTown.buyNewAnimal(AnimalType.ZEBRA);
8.         capeTown.buyNewAnimal(AnimalType.OWL);
9.         capeTown.buyNewAnimal(AnimalType.COW);
10.        capeTown.buyNewAnimal(AnimalType.LION);
11.        capeTown.buyNewAnimal(AnimalType.BEAR);
12.    }
```

```

13.         System.out.println("\nZoo in Moscow, Russian Federation");
14.         Zoo moscow = new RussianZoo();
15.         moscow.buyNewAnimal(AnimalType.BEAR);
16.         moscow.buyNewAnimal(AnimalType.OWL);
17.         moscow.buyNewAnimal(AnimalType.COW);
18.         moscow.buyNewAnimal(AnimalType.LION);
19.         moscow.buyNewAnimal(AnimalType.ZEBRA);
20.     }
21. }
22.

```

### ***Owl.java***

```

1. package FactoryMethod;
2.
3. public class Owl implements Animal {
4.     @Override
5.     public String getType() {
6.         return "Owl";
7.     }
8. }
9.

```

### ***RussianBear.java***

```

1. package FactoryMethod;
2.
3. public class RussianBear extends Bear {
4.     @Override
5.     public String getType() {
6.         return "Bear from Russia";
7.     }
8. }
9.

```

### ***RussianCow.java***

```

1. package FactoryMethod;
2.
3. public class RussianCow extends Cow {
4.     @Override
5.     public String getType() {
6.         return "Cow from Russia";
7.     }
8. }
9.

```



## ***RussianLion.java***

```
1. package FactoryMethod;
2.
3. public class RussianLion extends Lion {
4.     @Override
5.     public String getType() {
6.         return "Lion from Russia";
7.     }
8. }
9.
```

## ***RussianOwl.java***

```
1. package FactoryMethod;
2.
3. public class RussianOwl extends Owl {
4.     @Override
5.     public String getType() {
6.         return "Owl from Russia";
7.     }
8. }
9.
```

## ***RussianZoo.java***

```
1. package FactoryMethod;
2.
3. public class RussianZoo extends Zoo {
4.     @Override
5.     protected Animal createAnimal(AnimalType type) {
6.         Animal animal = null;
7.
8.         switch (type) {
9.             case BEAR:
10.                animal = new RussianBear();
11.                break;
12.             case COW:
13.                animal = new RussianCow();
14.                break;
15.             case OWL:
16.                animal = new RussianOwl();
17.                break;
18.             case LION:
19.                animal = new RussianLion();
20.                break;
21.             case ZEBRA:
22.                animal = new AfricanZebra();
23.                break;
24.        }
```

```

25.         return animal;
26.     }
27. }
28.

```

## ***Zebra.java***

```

1. package FactoryMethod;
2.
3. public class Zebra implements Animal {
4.     @Override
5.     public String getType() {
6.         return "Zebra from Africa";
7.     }
8. }
9.

```

## ***Zoo.java***

```

1. package FactoryMethod;
2.
3. public abstract class Zoo {
4.     public void buyNewAnimal(AnimalType type) {
5.         Animal animal = createAnimal(type);
6.         System.out.println("You just bought " + animal.getType() + " at the zoo!");
7.     }
8.
9.     protected abstract Animal createAnimal(AnimalType type);
10. }
11.

```

## ***Prototype:***

### ***Life.java***

```

1. package Prototype;
2.
3. public class Life {
4.     private Type type;
5.
6.     public Life() {
7.     }
8.
9.     public Life(Type type) {
10.        this.type = type;
11.    }
12.
13.    public Life copy() {
14.        return new Life();
15.    }

```

```

16.
17.     public Type getType() {
18.         return type;
19.     }
20.
21.     public void setType(Type type) {
22.         this.type = type;
23.     }
24.
25.     @Override
26.     public String toString() {
27.         return "Life{" +
28.             "type=" + type +
29.             '}';
30.     }
31. }
32.

```

### *Type.java*

```

1. package Prototype;
2.
3. public enum Type {
4.     BACTERIA,
5.     FUNGUS,
6.     ANIMAL,
7.     PLANT
8. }
9.

```

### *MainPrototype.java*

```

1. package Prototype;
2.
3. public class MainPrototype {
4.     public static void main(String[] args) {
5.         Life bacteria = new Life(Type.BACTERIA);
6.         Life fungus = bacteria.copy();
7.         fungus.setType(Type.FUNGUS);
8.
9.         System.out.println(bacteria);
10.        System.out.println(fungus);
11.    }
12. }
13.

```

## Результат выполнения программы

```
Создан африканский сад!  
Посажен Цветок - Глориоза  
Посажено Дерево - Баобаб  
  
Создан европейский сад!  
Посажен Цветок - Роза  
Посажено Дерево - Дуб  
  
Создан домашний сад  
Посажен Цветок - Орхидея  
Вы не можете посадить дома дерево!
```

Рисунок 6.1 – Демонстрация работы программы Abstract Factory

```
Лев обычно живет в Саванна, весит примерно 150 и живет 12 лет.  
Белый медведь обычно живет в Арктика, весит примерно 400 и живет 25 лет.  
Обыкновенная лисица обычно живет в тундра, и степи, и леса разного типа, и пустыни и высокогорья, весит примерно 10 и живет 4 лет.
```

Рисунок 6.2 – Демонстрация работы программы Main Builder

```
Zoo in Cape Town, Republic of South Africa  
You just bought Zebra from Africa at the zoo!  
You just bought Owl from Africa at the zoo!  
You just bought Cow from Africa at the zoo!  
You just bought Lion from Africa at the zoo!  
You just bought Bear from Russia at the zoo!  
  
Zoo in Moscow, Russian Federation  
You just bought Bear from Russia at the zoo!  
You just bought Owl from Russia at the zoo!  
You just bought Cow from Russia at the zoo!  
You just bought Lion from Russia at the zoo!  
You just bought Zebra from Africa at the zoo!
```

Рисунок 6.3 – Демонстрация работы программы Factory Method

```
Life{type=BACTERIA}  
Life{type=FUNGUS}
```

Рисунок 6.4 – Демонстрация работы программы Prototype

## Практическая работа №7

### *Цель работы*

Тема: Реализация структурных паттернов проектирования.

Постановка задачи: Написать реализацию паттерна в соответствии с вариантом индивидуального задания Легковес, Заместитель.

### *Листинг программы*

#### *Flyweight:*

##### *FirstHuman.java*

```
1. package Flyweight;
2.
3. public class FirstHuman extends Human {
4.     public FirstHuman() {
5.         firstName = "Иван";
6.         lastName = "Николаев-Аксенов";
7.         age = 19;
8.     }
9.
10.    @Override
11.    public void getInfo() {
12.        System.out.println(firstName + ' ' + lastName + ", возраст " + age + "
    лет");
13.    }
14. }
15.
```

##### *FlyweightFactory.java*

```
1. package Flyweight;
2.
3. import java.util.HashMap;
4.
5. public class FlyweightFactory {
6.     private HashMap<Integer, Human> people = new HashMap<>();
7.
8.     public Human getHumanInfo(int number) {
9.         Human human = people.get(number);
10.        if (human == null) {
11.            switch (number) {
12.                case 1: {
13.                    human = new FirstHuman();
14.                    break;
15.                }
16.            }
17.        }
18.        return human;
19.    }
20. }
```

```

16.             case 2: {
17.                 human = new SecondHuman();
18.                 break;
19.             }
20.             case 3: {
21.                 human = new ThirdHuman();
22.                 break;
23.             }
24.         }
25.         people.put(number, human);
26.     }
27.     return human;
28. }
29. }
30.

```

## *Human.java*

```

1. package Flyweight;
2.
3. public abstract class Human {
4.     protected String firstName;
5.     protected String lastName;
6.     protected int age;
7.     public abstract void getInfo();
8. }
9.

```

## *MainFlyweight.java*

```

1. package Flyweight;
2.
3. public class MainFlyweight {
4.     public static void main(String[] args) {
5.         FlyweightFactory factory = new FlyweightFactory();
6.
7.         int[] peopleList = {1, 2, 3, 3, 2};
8.         for(int i: peopleList) {
9.             Human h = factory.getHumanInfo(i);
10.            h.getInfo();
11.        }
12.    }
13. }
14.

```

## *SecondHuman.java*

```
1. package Flyweight;
2.
3. public class SecondHuman extends Human {
4.     public SecondHuman() {
5.         firstName = "Иван";
6.         lastName = "Иванов";
7.         age = 23;
8.     }
9.
10.    @Override
11.    public void getInfo() {
12.        System.out.println(firstName + ' ' + lastName + ", возраст " + age + "
лет");
13.    }
14. }
```

## *ThirdHuman.java*

```
1. package Flyweight;
2.
3. public class ThirdHuman extends Human {
4.     public ThirdHuman() {
5.         firstName = "Петр";
6.         lastName = "Петров";
7.         age = 48;
8.     }
9.
10.    @Override
11.    public void getInfo() {
12.        System.out.println(firstName + ' ' + lastName + ", возраст " + age + "
лет");
13.    }
14. }
15.
```

## *Proxy:*

### *IUserChanger.java*

```
1. package Proxy;
2.
3. import Flyweight.Human;
4.
5. public interface IUserChanger {
6.     void changeName(User user, String name);
7.     void changeAge(User user, int age);
8. }
9.
```

## *MainProxy.java*

```
1. package Proxy;
2.
3. public class MainProxy {
4.     public static void main(String[] args) {
5.         User a = new User("Ivan", 19);
6.         User b = new User("Petr", 25);
7.         User c = new User("root", 0);
8.
9.         UserChanger userChanger = new UserChanger();
10.        ProxyUserChanger proxyUserChanger = new ProxyUserChanger();
11.
12.        userChanger.changeAge(a, 21);
13.        userChanger.changeName(b, "Pavel");
14.
15.        proxyUserChanger.changeName(c, "Ivan&Pavel account");
16.
17.        System.out.println(a);
18.        System.out.println(b);
19.        System.out.println(c);
20.    }
21. }
22.
```

## *ProxyUserChanger.java*

```
1. package Proxy;
2.
3. public class ProxyUserChanger implements IUserChanger {
4.     private UserChanger uc;
5.
6.     @Override
7.     public void changeName(User user, String name) {
8.         System.out.println("Proxy...");
9.         UserChangerInitializer();
10.        uc.changeName(user, name);
11.    }
12.
13.    @Override
14.    public void changeAge(User user, int age) {
15.        System.out.println("Proxy...");
16.        UserChangerInitializer();
17.        uc.changeAge(user, age);
18.    }
19.
20.    private void UserChangerInitializer() {
21.        if(uc == null)
```



```

22.         uc = new UserChanger();
23.     }
24. }
25.

```

## *User.java*

```

1. package Proxy;
2.
3. public class User {
4.     private String name;
5.     private int age;
6.
7.     public User(String name, int age) {
8.         this.name = name;
9.         this.age = age;
10.    }
11.    public String getName() {
12.        return name;
13.    }
14.
15.    public void setName(String name) {
16.        this.name = name;
17.    }
18.
19.    public int getAge() {
20.        return age;
21.    }
22.
23.    public void setAge(int age) {
24.        this.age = age;
25.    }
26.
27.    @Override
28.    public String toString() {
29.        return "Пользователь " + name + ", возраст " + age + " лет.";
30.    }
31. }
32.

```

## *UserChanger.java*

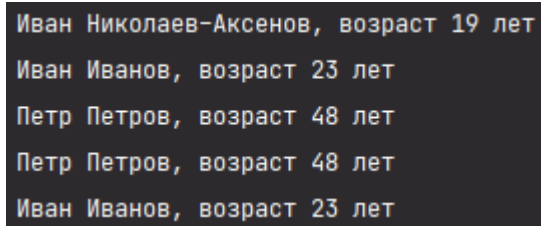
```

1. package Proxy;
2.
3. public class UserChanger implements IUserChanger {
4.     @Override
5.     public void changeName(User user, String name) {
6.         user.setName(name);
7.     }
8.

```

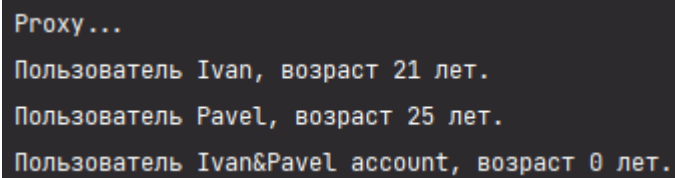
```
9.      @Override
10.     public void changeAge(User user, int age) {
11.         user.setAge(age);
12.     }
13. }
14.
```

### *Результат выполнения программы*



Иван Николаев-Аксенов, возраст 19 лет  
Иван Иванов, возраст 23 лет  
Петр Петров, возраст 48 лет  
Петр Петров, возраст 48 лет  
Иван Иванов, возраст 23 лет

Рисунок 7.1 – Демонстрация работы программы Flyweight



Прoxy...  
Пользователь Ivan, возраст 21 лет.  
Пользователь Pavel, возраст 25 лет.  
Пользователь Ivan&Pavel account, возраст 0 лет.

Рисунок 7.2 – Демонстрация работы программы Proxy

## Практическая работа №8

### *Цель работы*

Тема: Реализация поведенческих паттернов проектирования.

Постановка задачи: Написать реализацию паттерна в соответствии с вариантом индивидуального задания Наблюдатель, Состояние.

### *Листинг программы*

#### *Observer:*

##### *CollageRating.java*

```
1. package Observer;
2.
3. public class CollageRating implements Observer {
4.     private String name;
5.     private double gpa;
6.     public void printInfo() {
7.         System.out.println("Студент " + name + " имеет средний балл " + gpa);
8.     }
9.     @Override
10.    public void update(String name, double gpa) {
11.        this.name = name;
12.        this.gpa = gpa;
13.        printInfo();
14.    }
15. }
16.
```

##### *MainObserver.java*

```
1. package Observer;
2.
3. public class MainObserver {
4.     public static void main(String[] args) {
5.         Student student = new Student();
6.         Observer studentInfo = new CollageRating();
7.
8.         student.addObserver(studentInfo);
9.
10.        student.setNameAndGPA("Ivan", 7.9);
11.        student.setName("Petr");
12.        student.setGpa(5.2);
13.    }
14. }
15.
```

## *Observable.java*

```
1. package Observer;
2.
3. public interface Observable {
4.     void addObserver(Observer o);
5.     void removeObserver(Observer o);
6.     void notifyObservers();
7. }
8.
```

## *Observer.java*

```
1. package Observer;
2.
3. interface Observer {
4.     void update(String name, double gpa);
5. }
6.
```

## *Student.java*

```
1. package Observer;
2.
3. import java.util.LinkedList;
4. import java.util.List;
5.
6. public class Student implements Observable {
7.     private String name;
8.     private double gpa;
9.     private List<Observer> obs;
10.
11.     public Student() {
12.         obs = new LinkedList<>();
13.     }
14.
15.     public void setName(String name) {
16.         this.name = name;
17.         notifyObservers();
18.     }
19.
20.     public void setGpa(double gpa) {
21.         this.gpa = gpa;
22.         notifyObservers();
23.     }
24.
25.     public void setNameAndGPA(String name, double gpa) {
26.         this.name = name;
27.         this.gpa = gpa;
28.         notifyObservers();

```

```

29.     }
30.
31.     @Override
32.     public void addObserver(Observer o) {
33.         obs.add(o);
34.     }
35.
36.     @Override
37.     public void removeObserver(Observer o) {
38.         obs.remove(o);
39.     }
40.
41.     @Override
42.     public void notifyObservers() {
43.         for(Observer o: obs)
44.             o.update(name, gpa);
45.     }
46.
47.
48. }
49.

```

## ***State:***

### ***Bread.java***

```

1. package State;
2.
3. public class Bread implements State {
4.     private static final String name = "хлеб";
5.
6.     @Override
7.     public String getName() {
8.         return name;
9.     }
10.
11.     @Override
12.     public void make(StateInfo stateInfo) {
13.         stateInfo.setState(new SandwichWithButter());
14.     }
15.
16.     @Override
17.     public void eat(StateInfo stateInfo) {
18.         System.out.println("Сначала нужно приготовить бутерброд! Пока он на стадии:
19.         " + name);
20.     }
21.

```

## *HalfASandwich.java*

```
1. package State;
2.
3. public class HalfASandwich implements State {
4.     private static final String name = "половина бутерброда";
5.
6.     @Override
7.     public String getName() {
8.         return name;
9.     }
10.
11.    @Override
12.    public void make(StateInfo stateInfo) {
13.        System.out.println("Вы уже начали есть бутерброд!");
14.    }
15.
16.    @Override
17.    public void eat(StateInfo stateInfo) {
18.        System.out.println("Вы съели бутерброд");
19.    }
20. }
21.
```

## *MainState.java*

```
1. package State;
2.
3. public class MainState {
4.     public static void main(String[] args) {
5.         StateInfo stateInfo = new StateInfo();
6.
7.         stateInfo.make();
8.         System.out.println();
9.
10.        stateInfo.make();
11.        System.out.println();
12.
13.        stateInfo.make();
14.        System.out.println();
15.
16.        stateInfo.eat();
17.        System.out.println();
18.
19.        stateInfo.eat();
20.    }
21. }
22.
```

## ***SandwichWithButter.java***

```
1. package State;
2.
3. public class SandwichWithButter implements State {
4.     private static final String name = "хлеб с маслом";
5.
6.     @Override
7.     public String getName() {
8.         return name;
9.     }
10.
11.    @Override
12.    public void make(StateInfo stateInfo) {
13.        stateInfo.setState(new SandwichWithButterAndSausage());
14.    }
15.
16.    @Override
17.    public void eat(StateInfo stateInfo) {
18.        System.out.println("Сначала нужно приготовить бутерброд! Пока он на стадии:
19.        " + name);
20.    }
21.
```

## ***SandwichWithButterAndSausage.java***

```
1. package State;
2.
3. public class SandwichWithButterAndSausage implements State {
4.     private static final String name = "бутерброд";
5.
6.     @Override
7.     public String getName() {
8.         return name;
9.     }
10.
11.    @Override
12.    public void make(StateInfo stateInfo) {
13.        System.out.println("Вы уже приготовили бутерброд!");
14.    }
15.
16.    @Override
17.    public void eat(StateInfo stateInfo) {
18.        stateInfo.setState(new HalfASandwich());
19.    }
20. }
21.
```

## *State.java*

```
1. package State;
2.
3. public interface State {
4.     String getName();
5.     void make(StateInfo stateInfo);
6.     void eat(StateInfo stateInfo);
7. }
8.
```

## *StateInfo.java*

```
1. package State;
2.
3. public class StateInfo {
4.     private State state = new Bread();
5.
6.     public void make() {
7.         System.out.println("Готовим бутерброд, текущее состояние " +
8.             state.getName());
9.         state.make(this);
10.    }
11.
12.    public void eat() {
13.        System.out.println("Едим бутерброд, текущее состояние " + state.getName());
14.        state.eat(this);
15.    }
16.
17.    public void setState(State state) {
18.        System.out.println("Изменяем состояние бутерброда на " + state.getName());
19.        this.state = state;
20.    }
21.
22.    public State getState() {
23.        return state;
24.    }
25.
```



### *Результат выполнения программы*

```
Студент Ivan имеет средний балл 7.9  
Студент Petr имеет средний балл 7.9  
Студент Petr имеет средний балл 5.2
```

Рисунок 8.1 – Демонстрация работы программы Observer

```
Готовим бутерброд, текущее состояние хлеб  
Изменяем состояние бутерброда на хлеб с маслом  
  
Готовим бутерброд, текущее состояние хлеб с маслом  
Изменяем состояние бутерброда на бутерброд  
  
Готовим бутерброд, текущее состояние бутерброд  
Вы уже приготовили бутерброд!  
  
Едим бутерброд, текущее состояние бутерброд  
Изменяем состояние бутерброда на половина бутерброда  
  
Едим бутерброд, текущее состояние половина бутерброда  
Вы съели бутерброд
```

Рисунок 8.2 – Демонстрация работы программы State

## Практическая работа №9

### *Цель работы*

Тема: Знакомство с системой сборки приложения. Gradle.

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

### *Листинг программы*

#### *Main.java*

```
1. public class Main {
2.     public static void main(String[] args) {
3.         System.out.println("Hello World from Java from Gradle from Docker!");
4.     }
5. }
6.
```

#### *build.gradle*

```
1. plugins {
2.     id 'java'
3. }
4.
5. version '1.0'
6.
7. repositories {
8.     mavenCentral()
9. }
10.
11.jar {
12.    manifest {
13.        attributes(
14.            'Class-Path': configurations.compile.collect { it.getName()
15.                }.join(' '),
16.            'Main-Class': 'Main'
17.        )
18.    }
19.
20.task toDocker {
21.    doLast {
22.        def stdout = new ByteArrayOutputStream()
23.
```

```

24.         println("Moving jar-file...")
25.         ant.move file: "${buildDir}/libs/HelloWorldProject-1.0.jar", todir:
           "${projectDir}/Docker"
26.
27.         println("\nCreating Docker container...")
28.         exec {
29.             workingDir "${projectDir}/Docker"
30.             commandLine 'docker', 'build', '-t', 'helloworld', '.'
31.         }
32.
33.         println("\nLaunching Docker container...")
34.         exec {
35.             workingDir "${projectDir}/Docker"
36.             commandLine 'docker', 'run', 'helloworld'
37.             standardOutput = stdout
38.         }
39.         println "\nOutput from Docker container:\n${stdout}"
40.     }
41. }
42.
43. build.finalizedBy(toDocker)
44.

```

## ***Dockerfile***

```

1. FROM openjdk:11.0.10
2. WORKDIR /
3. ADD HelloWorldProject-1.0.jar HelloWorldProject-1.0.jar
4. EXPOSE 8080
5. CMD ["java", "-jar", "HelloWorldProject-1.0.jar"]
6.

```

## Результат выполнения программы

```
frisc@FRISCHMANN-PC C:\Users\frisc\GoogleDrive\Study\JavaPatterns\Практическая работа 9
$ gradle build

> Task :toDocker
Moving jar-file...

Creating Docker container...
#1 [internal] load build definition from Dockerfile
#1 sha256:6c5c8e382d5ffc99fd11fe872beaf043be1ed3e70ef41bbb510364459de397b4
#1 transferring dockerfile: 32B 0.0s done
#1 DONE 0.0s

#2 [internal] load .dockerignore
#2 sha256:25102dd82253e6e3bb110ffb279cf849b80ba191760989a2482b6c73cb5034ef
#2 transferring context: 2B done
#2 DONE 0.0s

#3 [internal] load metadata for docker.io/library/openjdk:11.0.10
#3 sha256:c791bb7f23fefe5c5adff6530397c072a994ba739566a87564fdc5351fa2c3f7
#3 ...

#4 [auth] library/openjdk:pull token for registry-1.docker.io
#4 sha256:ed3a1316321d3728cfff11008cf06c3b1efca30e4b130ef743d80f105fbad572
#4 DONE 0.0s

#3 [internal] load metadata for docker.io/library/openjdk:11.0.10
#3 sha256:c791bb7f23fefe5c5adff6530397c072a994ba739566a87564fdc5351fa2c3f7
#3 DONE 2.1s

#6 [internal] load build context
#6 sha256:2179e4b2d6821d2188e14451ec4b9d7241a230ff747224d91390bc498029a626
#6 transferring context: 786B 0.0s done
#6 DONE 0.0s

#5 [1/3] FROM docker.io/library/openjdk:11.0.10@sha256:d112a532f300ce7d35f5cd196c22fced33d0a28a7dd0227
019da5fb430528428
#5 sha256:b3762d9b4b5c25472a7d73be65d92352989745a5defee86b0dbc82b6a2dc2fa2
#5 CACHED

#7 [2/3] ADD HelloWorldProject-1.0.jar HelloWorldProject-1.0.jar
#7 sha256:21d9bb277dcde7bd35d595c59612d099980ae7377089eb4213faa46ac864a901
#7 DONE 0.0s

#8 exporting to image
#8 sha256:e8c613e07b0b7ff33893b694f7759a10d42e180f2b4dc349fb57dc6b71dcab00
#8 exporting layers 0.0s done
#8 writing image sha256:077d11fbff279741454c3bc89476399fed405787a6639361d6cb6b1a59c5cc14
#8 writing image sha256:077d11fbff279741454c3bc89476399fed405787a6639361d6cb6b1a59c5cc14 done
#8 naming to docker.io/library/helloworld done
#8 DONE 0.0s

Launching Docker container...
```

Рисунок 9.1 – Демонстрация работы программы

```
Launching Docker container...

Output from Docker container:
Hello World from Java from Gradle from Docker!

Deprecated Gradle features were used in this build, making it incompatible with Gradle 7.0.
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/6.8/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 6s
3 actionable tasks: 2 executed, 1 up-to-date
```

Рисунок 9.2 – Демонстрация работы программы

## Практическая работа №10

### *Цель работы*

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

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

### *Листинг программы*

#### *Boxer.java*

```
1. package App;
2.
3. import org.springframework.stereotype.Component;
4.
5. @Component
6. public class Boxer implements Fighter {
7.     @Override
8.     public void doFight() {
9.         System.out.println("Boxer enters the ring");
10.    }
11. }
12.
```

#### *Fighter.java*

```
1. package App;
2.
3. public interface Fighter {
4.     void doFight();
5. }
6.
```

## *Judoka.java*

```
1. package App;
2.
3. import org.springframework.stereotype.Component;
4.
5. @Component
6. public class Judoka implements Fighter {
7.     @Override
8.     public void doFight() {
9.         System.out.println("Judoka fighter enters the ring");
10.    }
11. }
12.
```

## *StreetFighter.java*

```
1. package App;
2.
3. import org.springframework.context.ApplicationContext;
4. import org.springframework.context.support.ClassPathXmlApplicationContext;
5.
6. public class MainApp {
7.     public static void main(String[] args) {
8.         ApplicationContext ac = new ClassPathXmlApplicationContext("fighters.xml");
9.         Fighter f = (Fighter) ac.getBean(args[0]);
10.        f.doFight();
11.    }
12. }
13.
```

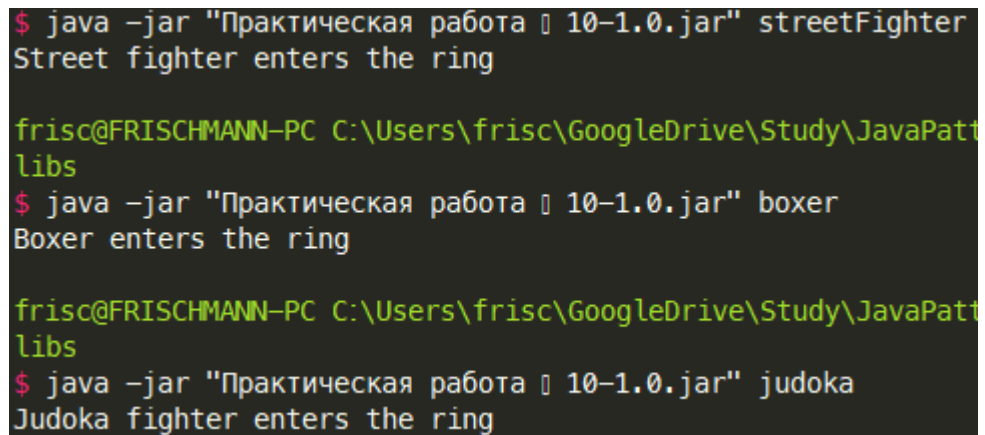
## *MainApp.java*

```
1. package App;
2.
3. import org.springframework.stereotype.Component;
4.
5. @Component
6. public class StreetFighter implements Fighter {
7.     @Override
8.     public void doFight() {
9.         System.out.println("Street fighter enters the ring");
10.    }
11. }
12.
```

## *fighters.xml*

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.       xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans.xsd">
5.     <bean id="streetFighter" class="App.StreetFighter"/>
6.     <bean id="boxer" class="App.Boxer"/>
7.     <bean id="judoka" class="App.Judoka"/>
8. </beans>
9.
```

## *Результат выполнения программы*



```
$ java -jar "Практическая работа 10-1.0.jar" streetFighter
Street fighter enters the ring

frisc@FRISCHMANN-PC C:\Users\frisc\GoogleDrive\Study\JavaPattern\libs
$ java -jar "Практическая работа 10-1.0.jar" boxer
Boxer enters the ring

frisc@FRISCHMANN-PC C:\Users\frisc\GoogleDrive\Study\JavaPattern\libs
$ java -jar "Практическая работа 10-1.0.jar" judoka
Judoka fighter enters the ring
```

Рисунок 10.1 – Демонстрация работы программы

## Практическая работа №11

### *Цель работы*

Тема: Разобраться с использованием Spring boot.

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

- Spring Web;
- Lombok;
- Validation;
- Spring boot Actuator.

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

### *Листинг программы*

#### *TestRest.java*

```
1. package App.AppMain;
2.
3. import org.springframework.web.bind.annotation.RequestMapping;
4. import org.springframework.web.bind.annotation.RestController;
5.
6. @RestController
7. public class TestRest {
8.     @RequestMapping("/test")
9.     public String open() {
10.         return "Hello, World!";
11.     }
12. }
13.
```

#### *AppMainApplication.java*

```
1. package App.AppMain;
2.
3. import org.springframework.boot.SpringApplication;
4. import org.springframework.boot.autoconfigure.SpringBootApplication;
5.
6. @SpringBootApplication
7. public class AppMainApplication {
8.
9.     public static void main(String[] args) {
10.         SpringApplication.run(AppMainApplication.class, args);
11.     }
12. }
```



```
11.  
12.         TestRest tr = new TestRest();  
13.         tr.open();  
14.     }  
15. }  
16.
```

## *Application.properties*

```
1. management.endpoint.auditevents.enabled=true  
2. management.endpoint.beans.enabled=true  
3. management.endpoint.caches.enabled=true  
4. management.endpoint.conditions.enabled=true  
5. management.endpoint.configprops.enabled=true  
6. management.endpoint.env.enabled=true  
7. management.endpoint.flyway.enabled=true  
8. management.endpoint.health.enabled=true  
9. management.endpoint.httptrace.enabled=true  
10. management.endpoint.info.enabled=true  
11. management.endpoint.integrationgraph.enabled=true  
12. management.endpoint.loggers.enabled=true  
13. management.endpoint.liquibase.enabled=true  
14. management.endpoint.metrics.enabled=true  
15. management.endpoint.mappings.enabled=true  
16. management.endpoint.scheduledtasks.enabled=true  
17. management.endpoint.sessions.enabled=true  
18. management.endpoint.shutdown.enabled=true  
19. management.endpoint.threaddump.enabled=true
```

## Результат выполнения программы

```

      ____          _            __ _ _
     /  _ \        | |          // (_) |
    / _ \| |   ___| | ___   ___//__|_| |_| |   _/_/___) (___) |
   / ___ \| |__/ __| |/ _ \ / __//__|_| |_| |   /___)\___) |
  /_/ ___\___/\___|_|\___/\___//__|_| |_| |_|_|___|_|___) |

:: Spring Boot ::                (v2.4.4)

2021-03-23 00:06:44.384 INFO 13560 --- [main] App.AppMain.AppMainApplication : Starting AppMainApplication using Java 11.0.10 on Frischmann-PC with PID 13560
2021-03-23 00:06:44.387 INFO 13560 --- [main] App.AppMain.AppMainApplication : No active profile set, falling back to default profiles: default
2021-03-23 00:06:45.851 INFO 13560 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2021-03-23 00:06:45.866 INFO 13560 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2021-03-23 00:06:45.867 INFO 13560 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.44]
2021-03-23 00:06:45.971 INFO 13560 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2021-03-23 00:06:45.971 INFO 13560 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1528 ms
2021-03-23 00:06:46.248 INFO 13560 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2021-03-23 00:06:46.526 INFO 13560 --- [main] o.s.b.a.e.web.EndpointLinksResolver : Exposing 2 endpoint(s) beneath base path '/actuator'
2021-03-23 00:06:46.573 INFO 13560 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-03-23 00:06:46.587 INFO 13560 --- [main] App.AppMain.AppMainApplication : Started AppMainApplication in 2.639 seconds (JVM running for 3.913)
2021-03-23 00:06:48.044 INFO 13560 --- [1]-172.29.128.1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2021-03-23 00:06:48.044 INFO 13560 --- [1]-172.29.128.1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2021-03-23 00:06:48.046 INFO 13560 --- [1]-172.29.128.1] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms

```

Рисунок 11.1 – Демонстрация работы программы

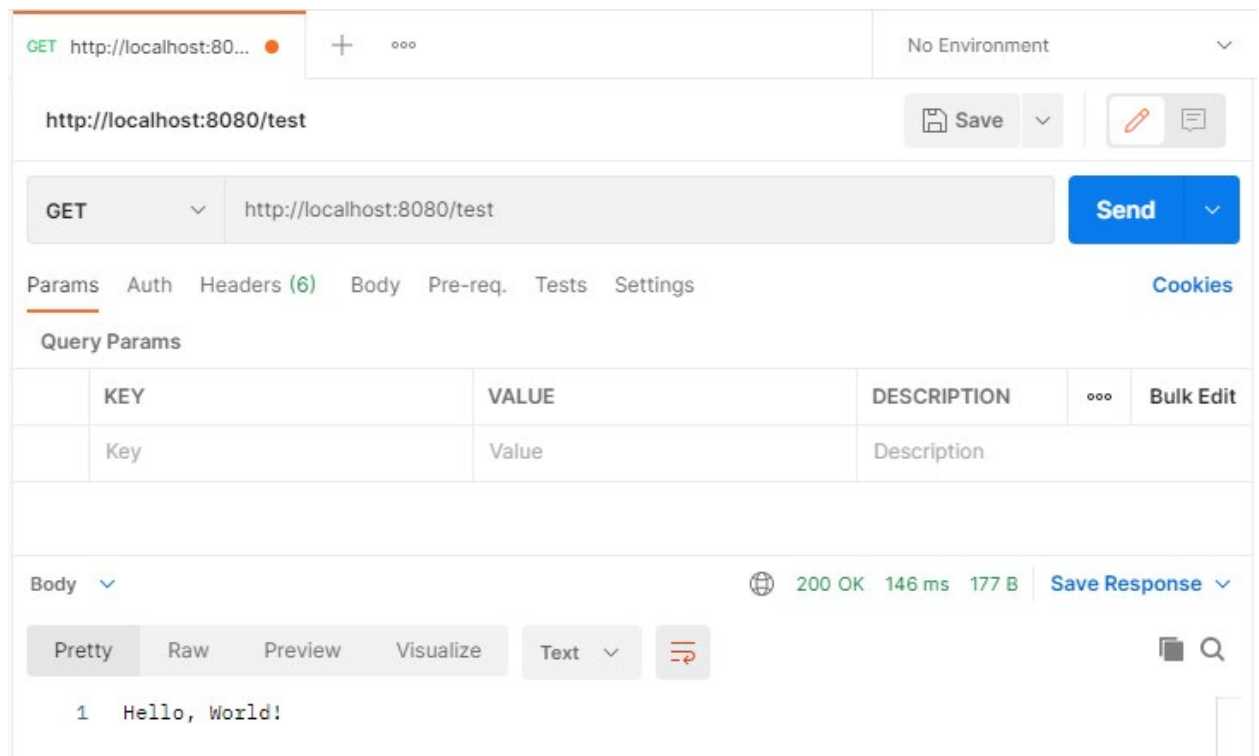


Рисунок 11.2 – Демонстрация работы программы

## Практическая работа №12

### *Цель работы*

Тема: Работа с жизненным циклом компонентов. Аннотации PostConstruct, PreDestroy.

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

### *Листинг программы*

#### *FileWorker.java*

```
1. package App.Main;
2.
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.boot.ApplicationArguments;
5. import org.springframework.stereotype.Component;
6. import org.springframework.util.DigestUtils;
7.
8. import javax.annotation.PostConstruct;
9. import javax.annotation.PreDestroy;
10. import java.io.*;
11.
12. @Component
13. public class FileWorker {
14.     @Autowired
15.     private ApplicationArguments arguments;
16.
17.     private String hashed;
18.
19.     public FileWorker() {
20.         hashed = "";
21.     }
22.
23.     @PostConstruct
24.     public void init() throws IOException {
25.         try(InputStream file = new
26.             FileInputStream(arguments.getNonOptionArgs().get(0))) {
27.             hashed = DigestUtils.md5DigestAsHex(file);
28.
29.             File secondFile = new File(arguments.getNonOptionArgs().get(1));
30.             if(!secondFile.exists())
```

```

30.         secondFile.createNewFile();
31.
32.         FileWriter writer = new FileWriter(secondFile);
33.         writer.write(hash);
34.         writer.close();
35.
36.     } catch (FileNotFoundException e) {
37.         File file = new File(arguments.getNonOptionArgs().get(1));
38.         if(!file.exists())
39.             file.createNewFile();
40.
41.         FileWriter writer = new FileWriter(file);
42.         writer.write("null");
43.         writer.close();
44.     } catch (IOException e) {
45.         e.printStackTrace();
46.     }
47. }
48. @PreDestroy
49. public void deleteFirst(){
50.     File file = new File(arguments.getNonOptionArgs().get(0));
51.
52.     if(file.exists()) {
53.         file.delete();
54.     }
55. }
56. }
57.

```

## *MainApplication.java*

```

1. package App.Main;
2.
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.boot.SpringApplication;
5. import org.springframework.boot.autoconfigure.SpringBootApplication;
6.
7. @SpringBootApplication
8. public class MainApplication {
9.     private final FileWorker worker;
10.
11.     @Autowired
12.     public MainApplication(FileWorker worker) {
13.         this.worker = worker;
14.     }
15.     public static void main(String[] args) throws Exception {
16.         SpringApplication.run(MainApplication.class, args);
17.     }
18. }

```

## *Результат выполнения программы*

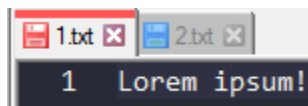


Рисунок 12.1 – Демонстрация работы программы

2.txt	23.03.2021 0:09	Файл "TXT"	1 КБ
Main-0.0.1.jar	22.03.2021 1:21	Файл "JAR"	8 360 КБ

Рисунок 12.2 – Демонстрация работы программы

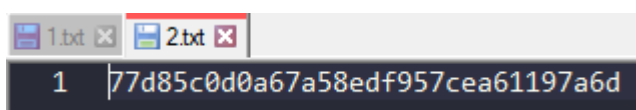


Рисунок 12.3 – Демонстрация работы программы

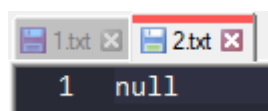


Рисунок 12.4 – Демонстрация работы программы

## Практическая работа №13

### *Цель работы*

Тема: Конфигурирование приложения. Environment.

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

- student.name – имя студента;
- student.last\_name – фамилия студента;
- student.group – название группы студента.

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

### *Листинг программы*

#### *Student.java*

```
1. package App.Main;
2.
3. import org.springframework.beans.factory.annotation.Value;
4. import org.springframework.stereotype.Component;
5.
6. import javax.annotation.PostConstruct;
7.
8. @Component
9. public class Student {
10.     @Value("${student.name}")
11.     private String name;
12.
13.     @Value("${student.last_name}")
14.     private String last_name;
15.
16.     @Value("${student.group}")
17.     private String group;
18.
19.     @PostConstruct
20.     public void init() {
21.         System.out.println("First name: " + name + "\nLast name: " + last_name +
22.             "\nGroup: " + group);
23.     }
24. }
```

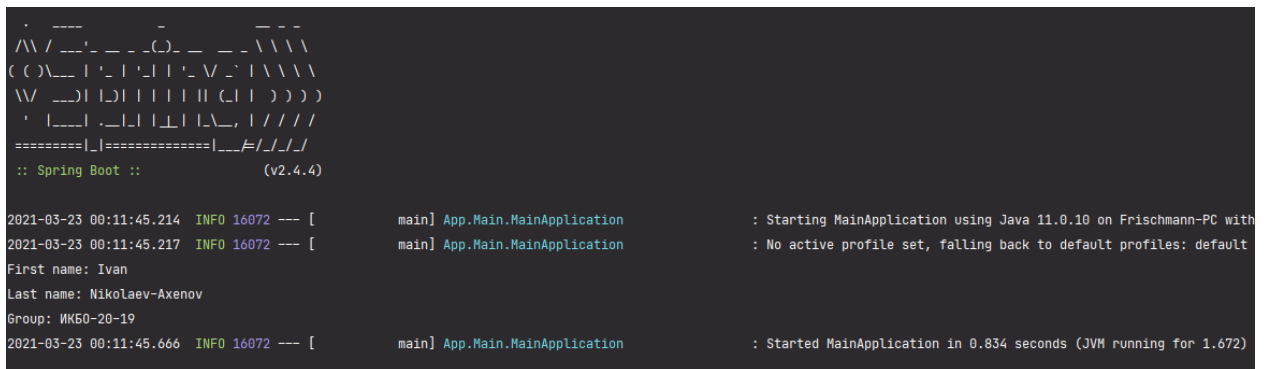
## *MainApplication.java*

```
1. package App.Main;
2.
3. import org.springframework.boot.SpringApplication;
4. import org.springframework.boot.autoconfigure.SpringBootApplication;
5.
6. @SpringBootApplication
7. public class MainApplication {
8.
9.     public static void main(String[] args) {
10.         SpringApplication.run(MainApplication.class, args);
11.     }
12. }
13.
```

## *application.yml*

```
1. student:
2.   name: Ivan
3.   last_name: Nikolaev-Axenov
4.   group: ИКБО-20-19
```

## *Результат выполнения программы*



```
.
 _ _ _ _ _
/ \ / _ _ _ _ _ \
( ( \ _ _ _ _ _ /
W _ _ _ _ _ \
' _ _ _ _ _ \
=====|_|=====|_|_/././
:: Spring Boot ::      (v2.4.4)

2021-03-23 00:11:45.214 INFO 16072 --- [main] App.Main.MainApplication : Starting MainApplication using Java 11.0.10 on Frischmann-PC with
2021-03-23 00:11:45.217 INFO 16072 --- [main] App.Main.MainApplication : No active profile set, falling back to default profiles: default
First name: Ivan
Last name: Nikolaev-Axenov
Group: ИКБО-20-19
2021-03-23 00:11:45.666 INFO 16072 --- [main] App.Main.MainApplication : Started MainApplication in 0.834 seconds (JVM running for 1.672)
```

Рисунок 13.1 – Демонстрация работы программы

## Практическая работа №14

### *Цель работы*

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

Постановка задачи: Создать отдельный репозиторий Git. Создать простой html-документ, который будет содержать вашу фамилию, имя, номер группы, номер варианта. Создать контроллер, который будет возвращать данный статический документ при переходе на url «/home». Выполнить задание в зависимости с вариантом индивидуального задания Создать класс Post с полями text, creationDate. Создать класс User с полями firstName, lastName, middleName, birthDate. Создать классы-контроллеры для создания, удаления объектов и получения всех объектов каждого типа. Сами объекты хранить в памяти.

### *Листинг программы*

#### *Application.java*

Код не изменился с предыдущей задачи.

#### *Post.java*

```
1. package PR14.Application.model;
2.
3. import com.fasterxml.jackson.annotation.JsonProperty;
4.
5. import java.util.Date;
6.
7. public class Post {
8.     private final String text;
9.     private final Date creationDate;
10.
11.     public Post(@JsonProperty("text") String text) {
12.         this.text = text;
13.         this.creationDate = new Date();
14.     }
15.
16.     public String getText() {
17.         return text;
18.     }
19.
20.     public Date getCreationDate() {
21.         return creationDate;
22.     }
23.
```



```

24.     @Override
25.     public String toString() {
26.         return "Номер от " + creationDate + "\nТекст: " + text;
27.     }
28. }
29.

```

## *User.java*

```

1. package PR14.Application.model;
2.
3. import com.fasterxml.jackson.annotation.JsonProperty;
4.
5. import java.util.ArrayList;
6. import java.util.List;
7.
8. public class User {
9.     private final String firstName;
10.    private final String lastName;
11.    private final String middleName;
12.    private final String birthDate;
13.
14.    private final List<Post> posts = new ArrayList<>();
15.
16.    public User(@JsonProperty("firstName") String firstName,
17.               @JsonProperty("lastName") String lastName,
18.               @JsonProperty("middleName") String middleName,
19.               @JsonProperty("birthDate") String birthDate) {
20.        this.firstName = firstName;
21.        this.lastName = lastName;
22.        this.middleName = middleName;
23.        this.birthDate = birthDate;
24.    }
25.
26.    public String getFirstName() {
27.        return firstName;
28.    }
29.
30.    public String getLastName() {
31.        return lastName;
32.    }
33.
34.    public String getMiddleName() {
35.        return middleName;
36.    }
37.
38.    public String getBirthDate() {
39.        return birthDate;

```

```

40.     }
41.
42.     public List<Post> getPosts() {
43.         return posts;
44.     }
45.
46.     public void addPost(Post post) {
47.         this.posts.add(post);
48.     }
49.
50.     public void deletePost(Post post) {
51.         this.posts.remove(post);
52.     }
53.
54.     @Override
55.     public String toString() {
56.         return "Пользователь " + lastName + " " + firstName + " " + middleName + ",
    день рождения: " + birthDate + "\nОпубликовал следующие посты: " + posts;
57.     }
58. }
59.

```

### ***UserPostHolder.java***

```

1. package PR14.Application.model;
2.
3. import com.fasterxml.jackson.annotation.JsonProperty;
4.
5. public class UserPostHolder {
6.     private final User user;
7.     private final String text;
8.
9.     public UserPostHolder(@JsonProperty("user") User user,
10.        @JsonProperty("text") String text) {
11.         this.user = user;
12.         this.text = text;
13.     }
14.
15.     public User getUser() {
16.         return user;
17.     }
18.
19.     public String getText() {
20.         return text;
21.     }
22. }
23.

```

## *UserDataAccessService.java*

```
1. package PR14.Application.service;
2.
3. import PR14.Application.model.Post;
4. import PR14.Application.model.User;
5. import PR14.Application.model.UserPostHolder;
6. import org.springframework.stereotype.Repository;
7.
8. import java.util.ArrayList;
9. import java.util.List;
10.
11. @Repository
12. public class UserDataAccessService {
13.     private static List<User> DB = new ArrayList<>();
14.
15.     public int insertUser(User user) {
16.         DB.add(user);
17.         return 1;
18.     }
19.
20.     public int insertPost(UserPostHolder userPostHolder) {
21.         User user = userPostHolder.getUser();
22.         String text = userPostHolder.getText();
23.         for(User i : DB) {
24.             if(i.getFirstName().equals(user.getFirstName()) &&
25.                i.getLastName().equals(user.getLastName()) &&
26.                i.getMiddleName().equals(user.getMiddleName()) &&
27.                i.getBirthDate().equals(user.getBirthDate())) {
28.                 i.addPost(new Post(text));
29.             }
30.         }
31.         return 1;
32.     }
33.
34.     public int deleteUser(User user) {
35.         DB.removeIf(i -> i.getFirstName().equals(user.getFirstName()) &&
36.            i.getLastName().equals(user.getLastName()) &&
37.            i.getMiddleName().equals(user.getMiddleName()) &&
38.            i.getBirthDate().equals(user.getBirthDate()));
39.         return 1;
40.     }
41.
42.     public int deletePost(UserPostHolder userPostHolder) {
43.         User user = userPostHolder.getUser();
44.         String text = userPostHolder.getText();
45.         for(User i : DB) {
46.             if(i.getFirstName().equals(user.getFirstName()) &&
47.                i.getLastName().equals(user.getLastName()) &&
```

```

        i.getMiddleName().equals(user.getMiddleName()) &&
        i.getBirthDate().equals(user.getBirthDate())) {
41.            for(Post j : i.getPosts()) {
42.                if(j.getText().equals(text)) {
43.                    i.deletePost(j);
44.                }
45.            }
46.        }
47.    }
48.    return 1;
49. }
50.
51. public List<User> getAllUsers() {
52.     return DB;
53. }
54. }
55.

```

## *UserService.java*

```

1. package PR14.Application.service;
2.
3. import PR14.Application.model.User;
4. import PR14.Application.model.UserPostHolder;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.stereotype.Service;
7.
8. import java.util.List;
9.
10. @Service
11. public class UserService {
12.     private final UserDataAccessService userDataAccessService;
13.
14.     @Autowired
15.     public UserService(UserDataAccessService userDataAccessService) {
16.         this.userDataAccessService = userDataAccessService;
17.     }
18.
19.     public int insertUser(User user) {
20.         return userDataAccessService.insertUser(user);
21.     }
22.
23.     public int insertPost(UserPostHolder userPostHolder) {
24.         return userDataAccessService.insertPost(userPostHolder);
25.     }
26.
27.     public int deleteUser(User user) {
28.         return userDataAccessService.deleteUser(user);
29.     }
30.

```

```

31.     public int deletePost(UserPostHolder userPostHolder) {
32.         return userDataAccessService.deletePost(userPostHolder);
33.     }
34.
35.     public List<User> getAllUsers() {
36.         return userDataAccessService.getAllUsers();
37.     }
38. }
39.

```

## *HomeController.java*

```

1. package PR14.Application.controller;
2.
3. import org.springframework.http.MediaType;
4. import org.springframework.stereotype.Controller;
5. import org.springframework.web.bind.annotation.GetMapping;
6. import org.springframework.web.bind.annotation.ResponseBody;
7.
8. @Controller
9. public class HomeController {
10.     @GetMapping(value = "/home", produces = MediaType.TEXT_HTML_VALUE)
11.     @ResponseBody
12.     public String homePage() {
13.         return "<html>\n" +
14.             "<head><title>Home</title></head>\n" +
15.             "<body>\n" +
16.             "Фамилия: Николаев-Аксенов<br><hr>\nИмя: Иван<br><hr>\nНомер  
группы: ИКБО-20-19<br><hr>\nНомер варианта: 21(6)<hr>" +
17.             "</body>\n" +
18.             "</html>";
19.     }
20. }
21.

```

## *UserController.java*

```
1. package PR14.Application.controller;
2.
3. import PR14.Application.model.User;
4. import PR14.Application.model.UserPostHolder;
5. import PR14.Application.service.UserService;
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.web.bind.annotation.*;
8.
9. import java.util.List;
10.
11. @RestController
12. public class UserController {
13.     private final UserService userService;
14.
15.     @Autowired
16.     public UserController(UserService userService) {
17.         this.userService = userService;
18.     }
19.
20.     @PostMapping("/users")
21.     public int insertUser(@RequestBody User user) {
22.         return userService.insertUser(user);
23.     }
24.
25.     @PostMapping("/posts")
26.     public int insertPost(@RequestBody UserPostHolder userPostHolder) {
27.         return userService.insertPost(userPostHolder);
28.     }
29.
30.     @DeleteMapping("/users")
31.     public int deleteUser(@RequestBody User user) {
32.         return userService.deleteUser(user);
33.     }
34.
35.     @DeleteMapping("/posts")
36.     public int deletePost(@RequestBody UserPostHolder userPostHolder) {
37.         return userService.deletePost(userPostHolder);
38.     }
39.
40.     @GetMapping("/users")
41.     public List<User> getAllUsers() {
42.         return userService.getAllUsers();
43.     }
44. }
45.
```

### Результат выполнения программы

[illegible]

Рисунок 14.1 – Демонстрация работы программы

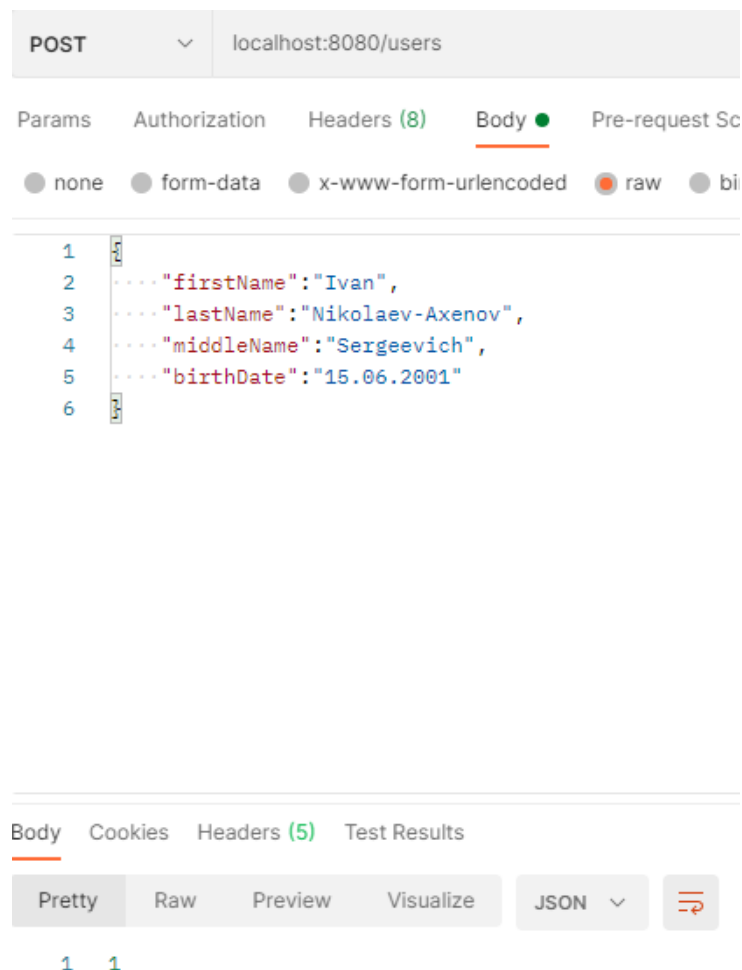


Рисунок 14.2 – Демонстрация работы программы



Рисунок 14.3 – Демонстрация работы программы

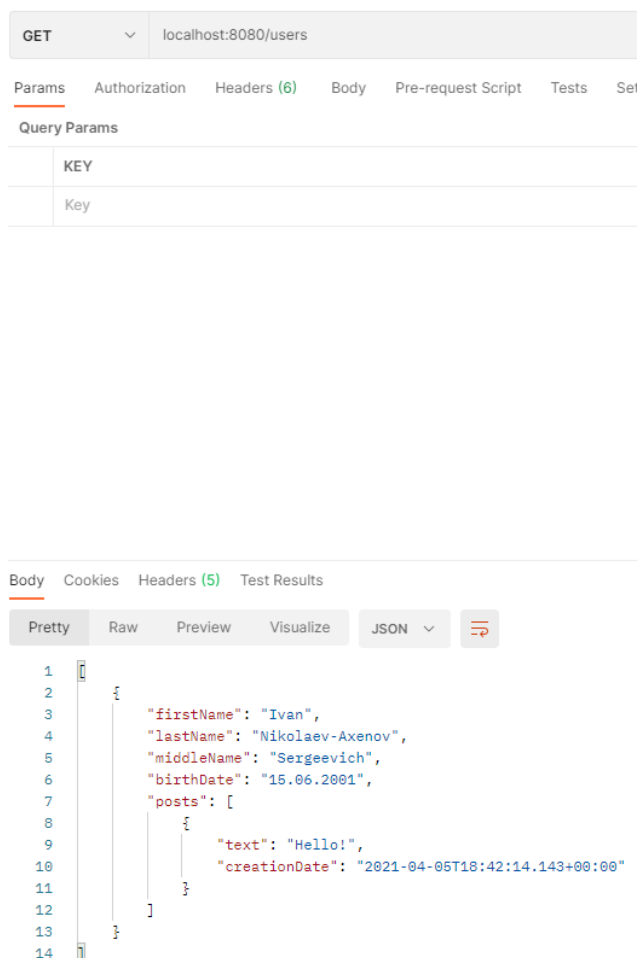


Рисунок 14.4 – Демонстрация работы программы



## Практическая работа №15

### *Цель работы*

Тема: Использование Hibernate в Spring framework.

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

### *Листинг программы*

#### *Application.java*

Код не изменился с предыдущей задачи.

#### *User.java*

```
1. package PR15.Application.model;
2.
3. import com.sun.istack.NotNull;
4. import org.hibernate.annotations.GenericGenerator;
5.
6. import javax.persistence.*;
7. import java.io.Serializable;
8. import java.util.UUID;
9.
10. @Entity
11. @Table(name = "users")
12. public class User implements Serializable {
13.     @Id
14.     @GeneratedValue(generator = "UUID")
15.     @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")
16.     @Column(name = "id", updatable = false, nullable = false)
17.     private UUID id;
18.
19.     @Column(name = "last_name")
20.     @NotNull
21.     private String lastName;
22.
23.     @Column(name = "first_name")
24.     @NotNull
25.     private String firstName;
26.
27.     @Column(name = "middle_name")
28.     @NotNull
29.     private String middleName;
30.
31.     @Column(name = "birth_date")
```

```

32.     @NotNull
33.     private String birthDate;
34.
35.     public User() {
36.
37.     }
38.
39.     public User(String lastName, String firstName, String middleName, String
        birthDate) {
40.         this.lastName = lastName;
41.         this.firstName = firstName;
42.         this.middleName = middleName;
43.         this.birthDate = birthDate;
44.     }
45.
46.     public UUID getId() {
47.         return id;
48.     }
49.
50.     public String getLastName() {
51.         return lastName;
52.     }
53.
54.     public String getFirstName() {
55.         return firstName;
56.     }
57.
58.     public String getMiddleName() {
59.         return middleName;
60.     }
61.
62.     public String getBirthDate() {
63.         return birthDate;
64.     }
65.
66.     @Override
67.     public String toString() {
68.         return "Пользователь #" + id + " " + lastName + " " + firstName + " " +
            middleName + ", день рождения: " + birthDate;
69.     }
70. }
71.

```

## *Post.java*

```

1. package PR15.Application.model;
2.
3. import com.sun.istack.NotNull;
4. import org.hibernate.annotations.CreationTimestamp;
5. import org.hibernate.annotations.GenericGenerator;

```

```

6. import org.springframework.format.annotation.DateTimeFormat;
7.
8. import javax.persistence.*;
9. import java.time.LocalDateTime;
10. import java.util.Date;
11. import java.util.UUID;
12.
13. @Entity
14. @Table(name = "posts")
15. public class Post {
16.     @Id
17.     @GeneratedValue(generator = "UUID")
18.     @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")
19.     @Column(name = "id", updatable = false, nullable = false)
20.     private UUID id;
21.
22.     @Column(name = "text")
23.     @NotNull
24.     private String text;
25.
26.     @CreationTimestamp
27.     @Column(name = "creation_date")
28.     private LocalDateTime creationDate;
29.
30.     @Column(name = "owner")
31.     @NotNull
32.     private UUID owner;
33.
34.     public Post() {
35.
36.     }
37.
38.     public Post(String text, UUID owner) {
39.         this.text = text;
40.         this.owner = owner;
41.     }
42.
43.     public UUID getId() {
44.         return id;
45.     }
46.
47.     public String getText() {
48.         return text;
49.     }
50.
51.     public LocalDateTime getCreationDate() {
52.         return creationDate;
53.     }
54.
55.     public UUID getOwner() {

```

```

56.         return owner;
57.     }
58. }
59.

```

## ***UserService.java***

```

1. package PR15.Application.service;
2.
3. import PR15.Application.model.User;
4. import org.hibernate.Session;
5. import org.hibernate.SessionFactory;
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.stereotype.Service;
8. import org.springframework.web.bind.annotation.PostMapping;
9.
10. import javax.annotation.PostConstruct;
11. import javax.annotation.PreDestroy;
12. import java.util.List;
13. import java.util.UUID;
14.
15. @Service
16. public class UserService {
17.     @Autowired
18.     private final SessionFactory sessionFactory;
19.
20.     private Session session;
21.
22.     public UserService(SessionFactory sessionFactory) {
23.         this.sessionFactory = sessionFactory;
24.     }
25.
26.     @PostConstruct
27.     public void init() {
28.         session = sessionFactory.openSession();
29.     }
30.
31.     @PreDestroy
32.     public void unSession() {
33.         session.close();
34.     }
35.
36.     public void addUser(User user) {
37.         session.beginTransaction();
38.         session.saveOrUpdate(user);
39.         session.getTransaction().commit();
40.     }
41.
42.     public List<User> getUsers() {
43.         return session.createQuery("select u from User u", User.class).list();

```

```

44.     }
45.
46.     public User getUser(UUID id) {
47.         return session.createQuery("select p from User u where u.id = p.id = '" +
            id + "'", User.class).getSingleResult();
48.     }
49.
50.     public void deleteUser(UUID id) {
51.         session.beginTransaction();
52.
53.         User t = session.load(User.class, id);
54.         session.delete(t);
55.
56.         session.getTransaction().commit();
57.     }
58. }
59.

```

## *PostService.java*

```

1. package PR15.Application.service;
2.
3. import PR15.Application.model.Post;
4. import org.hibernate.Session;
5. import org.hibernate.SessionFactory;
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.stereotype.Service;
8.
9. import javax.annotation.PostConstruct;
10. import javax.annotation.PreDestroy;
11. import java.util.List;
12. import java.util.UUID;
13.
14. @Service
15. public class PostService {
16.     @Autowired
17.     private final SessionFactory sessionFactory;
18.
19.     private Session session;
20.
21.     public PostService(SessionFactory sessionFactory) {
22.         this.sessionFactory = sessionFactory;
23.     }
24.
25.     @PostConstruct
26.     public void init() {
27.         session = sessionFactory.openSession();
28.     }
29.
30.     @PreDestroy

```

```

31.     public void unSession() {
32.         session.close();
33.     }
34.     public void addPost(Post post) {
35.         session.beginTransaction();
36.         session.saveOrUpdate(post);
37.         session.getTransaction().commit();
38.     }
39.     public List<Post> getPosts() {
40.         return session.createQuery("select p from Post p", Post.class).list();
41.     }
42.     public List<Post> getPost(UUID id) {
43.         return session.createQuery("select p from Post p where p.id='" + id + "'",
Post.class).list();
44.     }
45.
46.     public void deletePosts(Post post) {
47.         session.beginTransaction();
48.
49.         List<Post> query = session.createQuery("select p from Post p where p.id =
'" + post.getId() + "'", Post.class).list();
50.         for (Post p : query) {
51.             session.delete(p);
52.         }
53.
54.         session.getTransaction().commit();
55.     }
56.
57.     public void deletePost(UUID id) {
58.         session.beginTransaction();
59.
60.         Post t = session.load(Post.class, id);
61.         session.delete(t);
62.
63.         session.getTransaction().commit();
64.     }
65. }

```

## *UserController.java*

```

1. package PR15.Application.controller;
2.
3. import PR15.Application.model.User;
4. import PR15.Application.service.UserService;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.web.bind.annotation.*;
7.
8. import java.util.List;
9. import java.util.UUID;
10.

```

```

11. @RestController
12. public class UserController {
13.     @Autowired
14.     private UserService userService;
15.
16.     @PostMapping("/users")
17.     public void addUser(@RequestBody User user) {
18.         userService.addUser(user);
19.     }
20.
21.     @GetMapping("/users")
22.     public List<User> getUsers() {
23.         return userService.getUsers();
24.     }
25.
26.     @GetMapping("/users/{id}")
27.     public User getUser(@PathVariable UUID id) {
28.         return userService.getUser(id);
29.     }
30.
31.     @DeleteMapping("/users/{id}")
32.     public void deleteUser(@PathVariable UUID id) {
33.         userService.deleteUser(id);
34.     }
35. }
36.

```

## *PostController.java*

```

1. package PR15.Application.controller;
2.
3. import PR15.Application.model.Post;
4. import PR15.Application.service.PostService;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.web.bind.annotation.*;
7.
8. import java.util.List;
9. import java.util.UUID;
10.
11. @RestController
12. public class PostController {
13.     @Autowired
14.     private PostService postService;
15.
16.     @PostMapping("/post")
17.     public void addPost(@RequestBody Post post) {
18.         postService.addPost(post);
19.     }
20.
21.     @GetMapping("/posts")

```

```

22.     public List<Post> getAll() {
23.         return postService.getPosts();
24.     }
25.
26.     @GetMapping("/post/{id}")
27.     public List<Post> getPost(@PathVariable UUID id) {
28.         return postService.getPost(id);
29.     }
30.
31.     @DeleteMapping("/post/{id}")
32.     public void deletePost(@PathVariable UUID id) {
33.         postService.deletePost(id);
34.     }
35. }
36.

```

## *Config.java*

```

1. package PR15.Application.config;
2.
3. import com.zaxxer.hikari.HikariConfig;
4. import com.zaxxer.hikari.HikariDataSource;
5. import org.springframework.context.annotation.Bean;
6. import org.springframework.context.annotation.Configuration;
7. import org.springframework.orm.hibernate5.HibernateTransactionManager;
8. import org.springframework.orm.hibernate5.LocalSessionFactoryBean;
9. import org.springframework.transaction.PlatformTransactionManager;
10.
11. import javax.sql.DataSource;
12. import java.util.Properties;
13.
14. @Configuration
15. public class Config {
16.     @Bean
17.     public HikariDataSource dataSource() {
18.         HikariConfig config = new HikariConfig();
19.         config.setJdbcUrl("jdbc:postgresql://localhost:5432/pr15db");
20.         config.setUsername("postgres");
21.         config.setPassword("secret");
22.         config.setDriverClassName("org.postgresql.Driver");
23.         return new HikariDataSource(config);
24.     }
25.
26.     @Bean
27.     public LocalSessionFactoryBean sessionFactory(DataSource dataSource) {
28.         LocalSessionFactoryBean factoryBean = new LocalSessionFactoryBean();
29.         factoryBean.setDataSource(dataSource);
30.         factoryBean.setPackagesToScan("PR15.Application");
31.         Properties properties = new Properties();

```





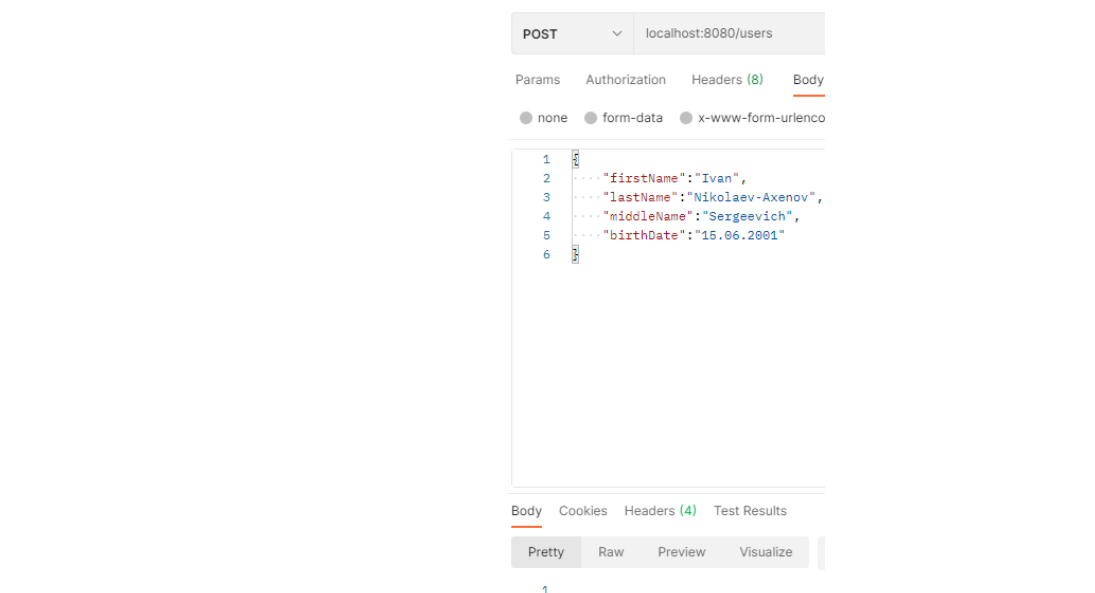


Рисунок 15.2 – Демонстрация работы программы



Рисунок 15.3 – Демонстрация работы программы

	id	last_name	first_name	middle_name	birth_date
1	c1e2e561-b3e2-4171-a8b8-e73be815d13f	Nikolaev-Axenov	Ivan	Sergeevich	15.06.2001
2	ada17906-0fb9-446b-b238-b05b05394d0b	Ivanov	Ivan	Ivanovich	02.02.2001

Рисунок 15.4 – Демонстрация работы программы

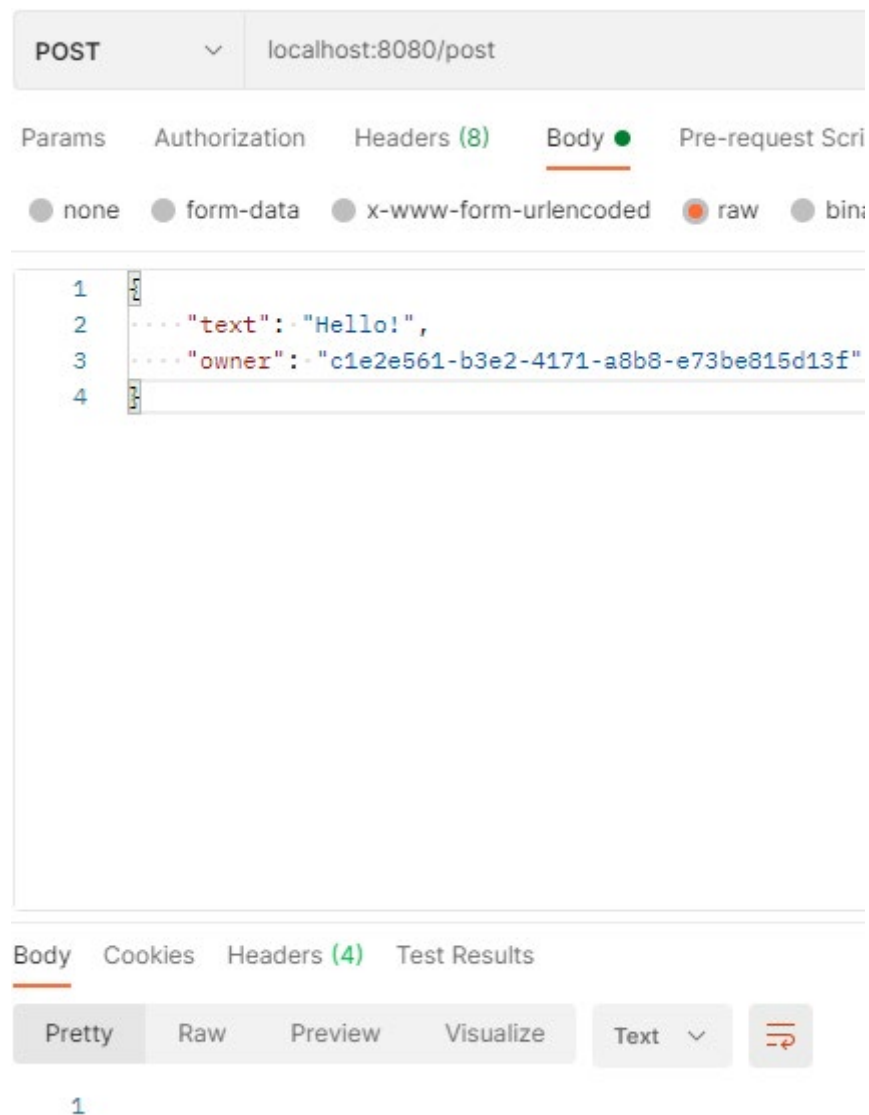


Рисунок 15.5 – Демонстрация работы программы

	id	text	creation_date	owner
1	aed67a07-663d-48eb-8520-0993f7b65019	Hello!	2021-04-05	c1e2e561-b3e2-4171-a8b8-e73be815d13f

Рисунок 15.6 – Демонстрация работы программы

## Практическая работа №16

### *Цель работы*

Тема: Изучение видов связей между сущностями в Hibernate. Использование транзакций.

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

### *Листинг программы*

#### *Application.java*

Код не изменился с предыдущей задачи.

#### *User.java*

```
1. package PR16.Application.model;
2.
3. import com.sun.istack.NotNull;
4. import org.hibernate.annotations.GenericGenerator;
5.
6. import javax.persistence.*;
7. import java.io.Serializable;
8. import java.util.*;
9.
10. @Entity
11. @Table(name = "users")
12. public class User implements Serializable {
13.     @Id
14.     @GeneratedValue(generator = "UUID")
15.     @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")
16.     @Column(name = "id", updatable = false, nullable = false)
17.     private UUID id;
18.
19.     @Column(name = "last_name")
20.     @NotNull
21.     private String lastName;
22.
23.     @Column(name = "first_name")
24.     @NotNull
25.     private String firstName;
26.
27.     @Column(name = "middle_name")
28.     @NotNull
29.     private String middleName;
```

```

30.
31.     @Column(name = "birth_date")
32.     @NotNull
33.     private String birthDate;
34.
35.     @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
36.     private List<Post> posts = new ArrayList<>();
37.
38.     public User() {
39.
40.     }
41.
42.     public User(String lastName, String firstName, String middleName, String
    birthDate) {
43.         this.lastName = lastName;
44.         this.firstName = firstName;
45.         this.middleName = middleName;
46.         this.birthDate = birthDate;
47.     }
48.
49.     public void addPost(Post post) {
50.         posts.add(post);
51.         post.setUser(this);
52.     }
53.
54.     public void removePost(Post post) {
55.         posts.remove(post);
56.         post.setUser(null);
57.     }
58.
59.     public UUID getId() {
60.         return id;
61.     }
62.
63.     public String getLastName() {
64.         return lastName;
65.     }
66.
67.     public String getFirstName() {
68.         return firstName;
69.     }
70.
71.     public String getMiddleName() {
72.         return middleName;
73.     }
74.
75.     public String getBirthDate() {
76.         return birthDate;
77.     }
78.

```

```

79.     @Override
80.     public String toString() {
81.         return "Пользователь #" + id + " " + lastName + " " + firstName + " " +
            middleName + ", день рождения: " + birthDate;
82.     }
83. }
84.

```

## *Post.java*

```

1. package PR16.Application.model;
2.
3. import com.sun.istack.NotNull;
4. import org.hibernate.annotations.CreationTimestamp;
5. import org.hibernate.annotations.GenericGenerator;
6. import org.springframework.format.annotation.DateTimeFormat;
7.
8. import javax.persistence.*;
9. import java.time.LocalDateTime;
10. import java.util.Date;
11. import java.util.UUID;
12.
13. @Entity
14. @Table(name = "posts")
15. public class Post {
16.     @Id
17.     @GeneratedValue(generator = "UUID")
18.     @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")
19.     @Column(name = "id", updatable = false, nullable = false)
20.     private UUID id;
21.
22.     @Column(name = "text")
23.     @NotNull
24.     private String text;
25.
26.     @CreationTimestamp
27.     @Column(name = "creation_date")
28.     private LocalDateTime creationDate;
29.
30.     @ManyToOne(fetch = FetchType.LAZY)
31.     private User user;
32.
33.     public Post() {
34.
35.     }
36.
37.     public Post(String text) {
38.         this.text = text;
39.     }
40.

```

```

41.     public void setUser(User user) {
42.         this.user = user;
43.     }
44.
45.     public UUID getId() {
46.         return id;
47.     }
48.
49.     public String getText() {
50.         return text;
51.     }
52.
53.     public LocalDateTime getCreationDate() {
54.         return creationDate;
55.     }
56. }
57.

```

## ***UserService.java***

```

1. package PR16.Application.service;
2.
3. import PR16.Application.model.Post;
4. import PR16.Application.model.User;
5. import org.hibernate.Session;
6. import org.hibernate.SessionFactory;
7. import org.springframework.beans.factory.annotation.Autowired;
8. import org.springframework.stereotype.Service;
9.
10. import javax.annotation.PostConstruct;
11. import javax.annotation.PreDestroy;
12. import java.util.List;
13. import java.util.UUID;
14.
15. @Service
16. public class UserService {
17.     @Autowired
18.     private final SessionFactory sessionFactory;
19.
20.     private Session session;
21.
22.     public UserService(SessionFactory sessionFactory) {
23.         this.sessionFactory = sessionFactory;
24.     }
25.
26.     @PostConstruct
27.     public void init() {
28.         session = sessionFactory.openSession();

```

```

29.     }
30.
31.     @PreDestroy
32.     public void unSession() {
33.         session.close();
34.     }
35.
36.     public void addUser(User user) {
37.         session.beginTransaction();
38.         session.saveOrUpdate(user);
39.         session.getTransaction().commit();
40.     }
41.
42.     public void addPost(UUID id, Post post) {
43.         session.beginTransaction();
44.
45.         User t = session.load(User.class, id);
46.         t.addPost(post);
47.         session.saveOrUpdate(t);
48.
49.         session.getTransaction().commit();
50.     }
51.
52.     public void removePost(UUID id, Post post) {
53.         session.beginTransaction();
54.
55.         User t = session.load(User.class, id);
56.         t.removePost(post);
57.         session.saveOrUpdate(t);
58.
59.         session.getTransaction().commit();
60.     }
61.
62.     public List<User> getUsers() {
63.         return session.createQuery("select u from User u", User.class).list();
64.     }
65.
66.     public User getUser(UUID id) {
67.         return session.createQuery("select u from User u where u.id = p.id = '" +
        id + "'", User.class).getSingleResult();
68.     }
69.
70.     public void deleteUser(UUID id) {
71.         session.beginTransaction();
72.         User t = session.load(User.class, id);
73.         session.delete(t);
74.
75.         session.getTransaction().commit();
76.     }
77. }

```



## *PostService.java*

```
1. package PR16.Application.service;
2.
3. import PR16.Application.model.Post;
4. import org.hibernate.Session;
5. import org.hibernate.SessionFactory;
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.stereotype.Service;
8.
9. import javax.annotation.PostConstruct;
10. import javax.annotation.PreDestroy;
11. import java.util.List;
12. import java.util.UUID;
13.
14. @Service
15. public class PostService {
16.     @Autowired
17.     private final SessionFactory sessionFactory;
18.
19.     private Session session;
20.
21.     public PostService(SessionFactory sessionFactory) {
22.         this.sessionFactory = sessionFactory;
23.     }
24.
25.     @PostConstruct
26.     public void init() {
27.         session = sessionFactory.openSession();
28.     }
29.
30.     @PreDestroy
31.     public void unSession() {
32.         session.close();
33.     }
34.
35.     public void addPost(Post post) {
36.         session.beginTransaction();
37.         session.saveOrUpdate(post);
38.         session.getTransaction().commit();
39.     }
40.
41.     public List<Post> getPosts() {
42.         return session.createQuery("select p from Post p", Post.class).list();
43.     }
44.
45.     public List<Post> getPost(UUID id) {
46.         return session.createQuery("select p from Post p where p.id='" + id + "'",
Post.class).list();
47.     }
```

```

48.
49.     public void deletePosts(Post post) {
50.         session.beginTransaction();
51.
52.         List<Post> query = session.createQuery("select p from Post p where p.id =
    '" + post.getId() + "'", Post.class).list();
53.         for (Post p : query) {
54.             session.delete(p);
55.         }
56.
57.         session.getTransaction().commit();
58.     }
59.
60.     public void deletePost(UUID id) {
61.         session.beginTransaction();
62.
63.         Post t = session.load(Post.class, id);
64.         session.delete(t);
65.
66.         session.getTransaction().commit();
67.     }
68. }
69.

```

## *UserController.java*

```

1. package PR16.Application.controller;
2.
3. import PR16.Application.model.Post;
4. import PR16.Application.model.User;
5. import PR16.Application.service.UserService;
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.web.bind.annotation.*;
8.
9. import java.util.List;
10. import java.util.UUID;
11.
12. @RestController
13. public class UserController {
14.     @Autowired
15.     private UserService userService;
16.
17.     @PostMapping("/users")
18.     public void addUser(@RequestBody User user) {
19.         userService.addUser(user);
20.     }
21.
22.     @GetMapping("/users")
23.     public List<User> getUsers() {
24.         return userService.getUsers();

```

```

25.     }
26.
27.     @PostMapping("/userpost/{id}")
28.     public void addPost(@PathVariable UUID id, @RequestBody String text) {
29.         userService.addPost(id, new Post(text));
30.     }
31.
32.     @DeleteMapping("/userpost/{id}")
33.     public void deletePost(@PathVariable UUID id, Post post) {
34.         userService.removePost(id, post);
35.     }
36.
37.     @GetMapping("/users/{id}")
38.     public User getUser(@PathVariable UUID id) {
39.         return userService.getUser(id);
40.     }
41.
42.     @DeleteMapping("/users/{id}")
43.     public void deleteUser(@PathVariable UUID id) {
44.         userService.deleteUser(id);
45.     }
46. }
47.

```

## *PostController.java*

```

1. package PR16.Application.controller;
2.
3. import PR16.Application.model.Post;
4. import PR16.Application.service.PostService;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.web.bind.annotation.*;
7.
8. import java.util.List;
9. import java.util.UUID;
10.
11. @RestController
12. public class PostController {
13.     @Autowired
14.     private PostService postService;
15.
16.     @PostMapping("/post")
17.     public void addPost(@RequestBody Post post) {
18.         postService.addPost(post);
19.     }
20.
21.     @GetMapping("/posts")
22.     public List<Post> getPosts() {
23.         return postService.getPosts();
24.     }

```



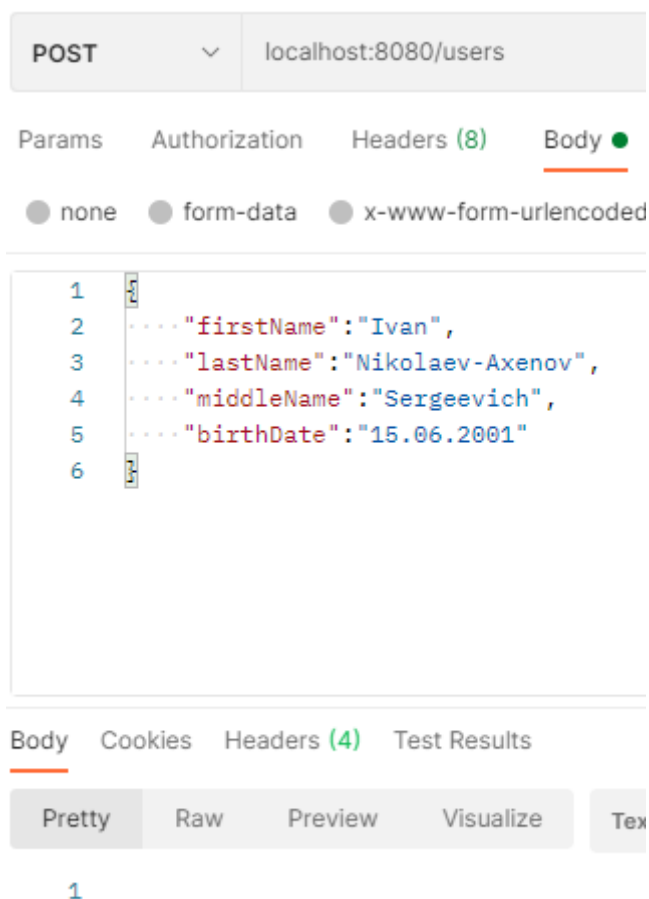


Рисунок 15.2 – Демонстрация работы программы

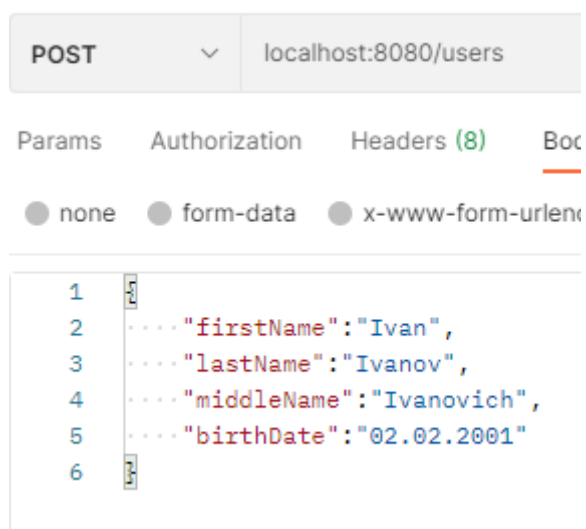


Рисунок 15.3 – Демонстрация работы программы

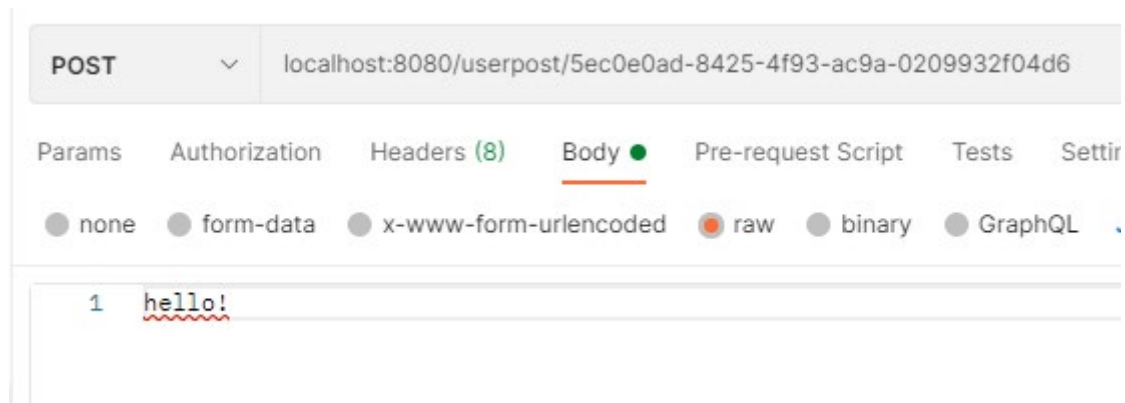


Рисунок 15.4 – Демонстрация работы программы

	id	text	creation_date	user_id
1	0a57064f-be5e-42e3-9c87-6a64bdbf527a	hello!	2021-04-05	5ec0e0ad-8425-4f93-ac9a-0209932f04d6

Рисунок 15.5 – Демонстрация работы программы

## Практическая работа №17

### *Цель работы*

Тема: Знакомство с Criteria API в Hibernate.

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

### *Листинг программы*

*application.java (в следующих работах тоже присутствует, но не изменяется)*

```
1. package app.Application;
2.
3. import org.springframework.boot.SpringApplication;
4. import org.springframework.boot.autoconfigure.SpringBootApplication;
5.
6. @SpringBootApplication
7. public class Application {
8.
9.     public static void main(String[] args) {
10.         SpringApplication.run(Application.class, args);
11.     }
12.
13. }
14.
```

### *User.java*

Код не изменился с предыдущей задачи.

### *Post.java*

Код не изменился с предыдущей задачи.

### *UserService.java*

```
1. package app.Application.service;
2.
3. import app.Application.model.User;
4. import org.hibernate.Session;
5. import org.hibernate.SessionFactory;
6. import org.hibernate.query.Query;
7. import org.springframework.beans.factory.annotation.Autowired;
```

```

8. import org.springframework.stereotype.Service;
9.
10. import javax.annotation.PostConstruct;
11. import javax.annotation.PreDestroy;
12. import javax.persistence.criteria.CriteriaBuilder;
13. import javax.persistence.criteria.CriteriaQuery;
14. import javax.persistence.criteria.Root;
15. import java.util.List;
16. import java.util.UUID;
17.
18. @Service
19. public class UserService {
20.     @Autowired
21.     private final SessionFactory sessionFactory;
22.     private Session session;
23.     private CriteriaBuilder builder;
24.     private CriteriaQuery<User> userCriteriaQuery;
25.     private Root<User> root;
26.
27.     public UserService(SessionFactory sessionFactory) {
28.         this.sessionFactory = sessionFactory;
29.     }
30.
31.     @PostConstruct
32.     public void init() {
33.         session = sessionFactory.openSession();
34.         builder = session.getCriteriaBuilder();
35.         userCriteriaQuery = builder.createQuery(User.class);
36.         root = userCriteriaQuery.from(User.class);
37.     }
38.
39.     @PreDestroy
40.     public void unSession() {
41.         session.close();
42.     }
43.
44.     public void addUser(User user) {
45.         session.beginTransaction();
46.         session.saveOrUpdate(user);
47.         session.getTransaction().commit();
48.     }
49.
50.     public List<User> getUsers() {
51.         return session.createQuery("select u from User u", User.class).list();
52.     }
53.
54.     public User getUser(UUID id) {
55.         return session.createQuery("select u from User u where u.id = p.id = '" +
56.             id + "'", User.class).getSingleResult();

```



```

57.
58.     public void deleteUser(UUID id) {
59.         session.beginTransaction();
60.
61.         User t = session.load(User.class, id);
62.         session.delete(t);
63.
64.         session.getTransaction().commit();
65.     }
66.
67.     public List<User> getByFirstName() {
68.         userCriteriaQuery.select(root).orderBy(builder.asc(root.get("firstName")));
69.         Query<User> query = session.createQuery(userCriteriaQuery);
70.         return query.getResultList();
71.     }
72.
73.     public List<User> getByLastName() {
74.         userCriteriaQuery.select(root).orderBy(builder.asc(root.get("lastName")));
75.         Query<User> query = session.createQuery(userCriteriaQuery);
76.         return query.getResultList();
77.     }
78. }

```

## *PostService.java*

```

1. package app.Application.service;
2.
3. import app.Application.model.Post;
4. import app.Application.model.User;
5. import org.hibernate.Session;
6. import org.hibernate.SessionFactory;
7. import org.hibernate.query.Query;
8. import org.springframework.beans.factory.annotation.Autowired;
9. import org.springframework.stereotype.Service;
10.
11. import javax.annotation.PostConstruct;
12. import javax.annotation.PreDestroy;
13. import javax.persistence.criteria.CriteriaBuilder;
14. import javax.persistence.criteria.CriteriaQuery;
15. import javax.persistence.criteria.Root;
16. import java.util.List;
17. import java.util.UUID;
18.
19. @Service
20. public class PostService {
21.     @Autowired
22.     private final SessionFactory sessionFactory;
23.
24.     private Session session;
25.     private CriteriaBuilder builder;

```

```

26.     private CriteriaQuery<Post> criteriaQuery;
27.     private Root<Post> root;
28.
29.     public PostService(SessionFactory sessionFactory) {
30.         this.sessionFactory = sessionFactory;
31.     }
32.
33.     @PostConstruct
34.     public void init() {
35.         session = sessionFactory.openSession();
36.         builder = session.getCriteriaBuilder();
37.         criteriaQuery = builder.createQuery(Post.class);
38.         root = criteriaQuery.from(Post.class);
39.     }
40.
41.     @PreDestroy
42.     public void unSession() {
43.         session.close();
44.     }
45.
46.     public void addPost(Post post) {
47.         session.beginTransaction();
48.         session.saveOrUpdate(post);
49.         session.getTransaction().commit();
50.     }
51.
52.     public List<Post> getPosts() {
53.         return session.createQuery("select p from Post p", Post.class).list();
54.     }
55.
56.     public User getUser(UUID id) {
57.         return session.createQuery("from Post where id = :id",
58.             Post.class).setParameter("id", id).getSingleResult().getUser();
59.     }
60.     public void deletePosts(Post post) {
61.         session.beginTransaction();
62.
63.         List<Post> query = session.createQuery("select p from Post p where p.id =
64.             " + post.getId() + "'", Post.class).list();
65.         for (Post p : query) {
66.             session.delete(p);
67.         }
68.         session.getTransaction().commit();
69.     }
70.
71.     public void deletePost(UUID id) {
72.         session.beginTransaction();
73.

```

```

74.         Post t = session.load(Post.class, id);
75.         session.delete(t);
76.
77.         session.getTransaction().commit();
78.     }
79.
80.     public List<Post> getByText() {
81.         criteriaQuery.select(root).orderBy(builder.asc(root.get("text")));
82.         Query<Post> query = session.createQuery(criteriaQuery);
83.         return query.getResultList();
84.     }
85.
86.     public List<Post> getByCreationDate() {
87.         criteriaQuery.select(root).orderBy(builder.asc(root.get("creationDate")));
88.         Query<Post> query = session.createQuery(criteriaQuery);
89.         return query.getResultList();
90.     }
91. }
92.

```

## *UserController.java*

```

1. package app.Application.controller;
2.
3. import app.Application.model.Post;
4. import app.Application.model.User;
5. import app.Application.service.UserService;
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.web.bind.annotation.*;
8.
9. import java.util.List;
10. import java.util.UUID;
11.
12. @RestController
13. public class UserController {
14.     @Autowired
15.     private UserService userService;
16.
17.     @PostMapping("/users")
18.     public void addUser(@RequestBody User user) {
19.         userService.addUser(user);
20.     }
21.
22.     @GetMapping("/users")
23.     public List<User> getUsers() {
24.         return userService.getUsers();
25.     }
26.
27.     @GetMapping("/users/{id}")
28.     public User getUser(@PathVariable UUID id) {

```

```

29.         return userService.getUser(id);
30.     }
31.
32.     @DeleteMapping("/users/{id}")
33.     public void deleteUser(@PathVariable UUID id) {
34.         userService.deleteUser(id);
35.     }
36.
37.     @GetMapping("/getByFirstName")
38.     public List<User> getByFirstName() {
39.         return userService.getByFirstName();
40.     }
41.
42.     @GetMapping("/getByLastName")
43.     public List<User> getByLastName() {
44.         return userService.getByLastName();
45.     }
46. }
47.

```

## *PostController.java*

```

1. package app.Application.controller;
2.
3. import app.Application.model.Post;
4. import app.Application.model.User;
5. import app.Application.service.PostService;
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.web.bind.annotation.*;
8.
9. import java.util.List;
10. import java.util.UUID;
11.
12. @RestController
13. public class PostController {
14.     @Autowired
15.     private PostService postService;
16.
17.     @PostMapping("/post")
18.     public void addPost(@RequestBody Post post) {
19.         postService.addPost(post);
20.     }
21.
22.     @GetMapping("/posts")
23.     public List<Post> getPosts() {
24.         return postService.getPosts();
25.     }
26.
27.     @DeleteMapping("/post/{id}")
28.     public void deletePost(@PathVariable UUID id) {

```

```

29.         postService.deletePost(id);
30.     }
31.
32.     @GetMapping("/getByText")
33.     public List<Post> getByText() {
34.         return postService.getByText();
35.     }
36.
37.     @GetMapping("/getByCreationDate")
38.     public List<Post> getByCreationDate() {
39.         return postService.getByCreationDate();
40.     }
41.
42.     @GetMapping(value = "/post/{id}/user")
43.     public @ResponseBody
44.     User getUser(@PathVariable("id") UUID id) {
45.         return postService.getUser(id);
46.     }
47. }
48.

```

### Config.java

Код не изменился с предыдущей задачи.

## Результат выполнения программы

```

  ____
 / __ \ / ___'  _ \ ( )_ _ _ _ _ \ \ \ \ \
( ( ( \___ \ ' _ \ ' _ \ V _ _ \ \ \ \ \
 \___ \ \___ \ \___ \ \___ \ \___ \ \___ \
  ' _ \ \___ \ \___ \ \___ \ \___ \ \___ \
=====|_|=====|_|_/_/_/_/_/

:: Spring Boot ::                (v2.4.4)

2021-04-10 15:09:36.311 INFO 9856 --- [main] app.Application.Application : Starting Application using Java 11.0.10 on Frischmann-PC with PID 9856
2021-04-10 15:09:36.314 INFO 9856 --- [main] app.Application.Application : No active profile set, falling back to default profiles: default
2021-04-10 15:09:37.458 INFO 9856 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2021-04-10 15:09:37.480 INFO 9856 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 11 ms. Found 0 JPA repositories
2021-04-10 15:09:38.169 INFO 9856 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2021-04-10 15:09:38.182 INFO 9856 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2021-04-10 15:09:38.182 INFO 9856 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.44]
2021-04-10 15:09:38.327 INFO 9856 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2021-04-10 15:09:38.328 INFO 9856 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1890 ms
2021-04-10 15:09:38.410 INFO 9856 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2021-04-10 15:09:38.626 INFO 9856 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2021-04-10 15:09:38.753 INFO 9856 --- [main] org.hibernate.Version : HHH0000412: Hibernate ORM core version 5.4.29.Final
2021-04-10 15:09:39.140 INFO 9856 --- [main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2021-04-10 15:09:39.374 INFO 9856 --- [main] org.hibernate.dialect.Dialect : HHH0000400: Using dialect: org.hibernate.dialect.PostgreSQLDialect
2021-04-10 15:09:40.968 INFO 9856 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH0000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2021-04-10 15:09:41.253 WARN 9856 --- [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may execute in the Spring application's context.
2021-04-10 15:09:41.365 INFO 9856 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2021-04-10 15:09:41.584 INFO 9856 --- [main] .s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: e34df85d-724d-4c59-9e67-4c9f4423cf6

2021-04-10 15:09:41.709 INFO 9856 --- [main] o.s.web.DefaultSecurityFilterChain : Will secure any request with [org.springframework.security.web.context.HttpSessionSecurityContextRepository, org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter]
2021-04-10 15:09:41.850 INFO 9856 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-04-10 15:09:41.861 INFO 9856 --- [main] app.Application.Application : Started Application in 8.508 seconds (JVM running for 10.932)

```

Рисунок 17.1 – Демонстрация работы программы

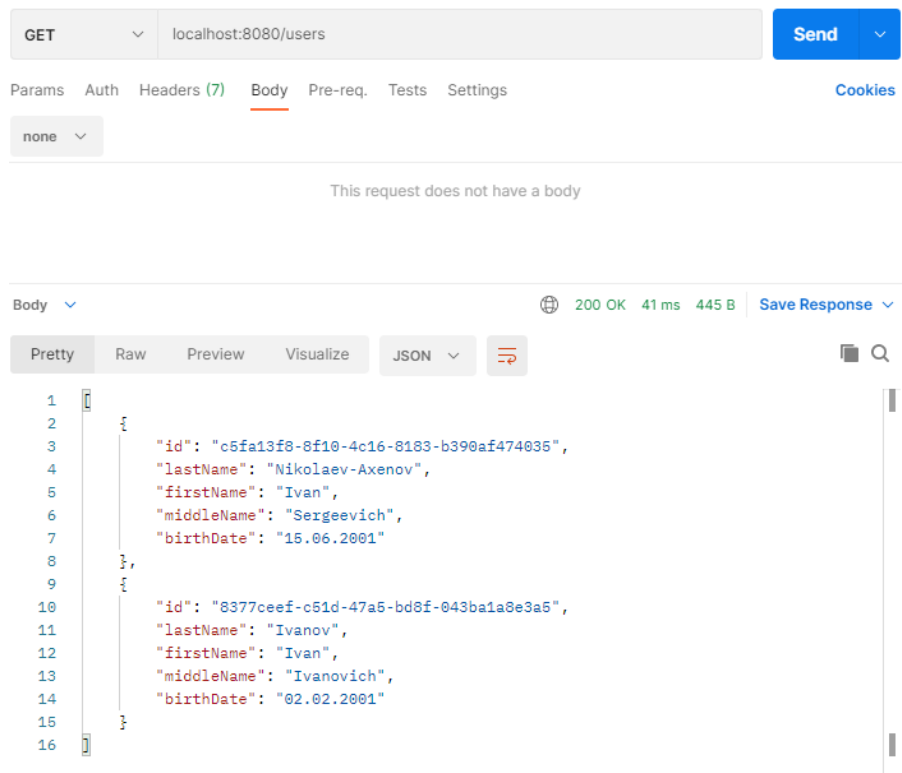


Рисунок 17.2 – Демонстрация работы программы

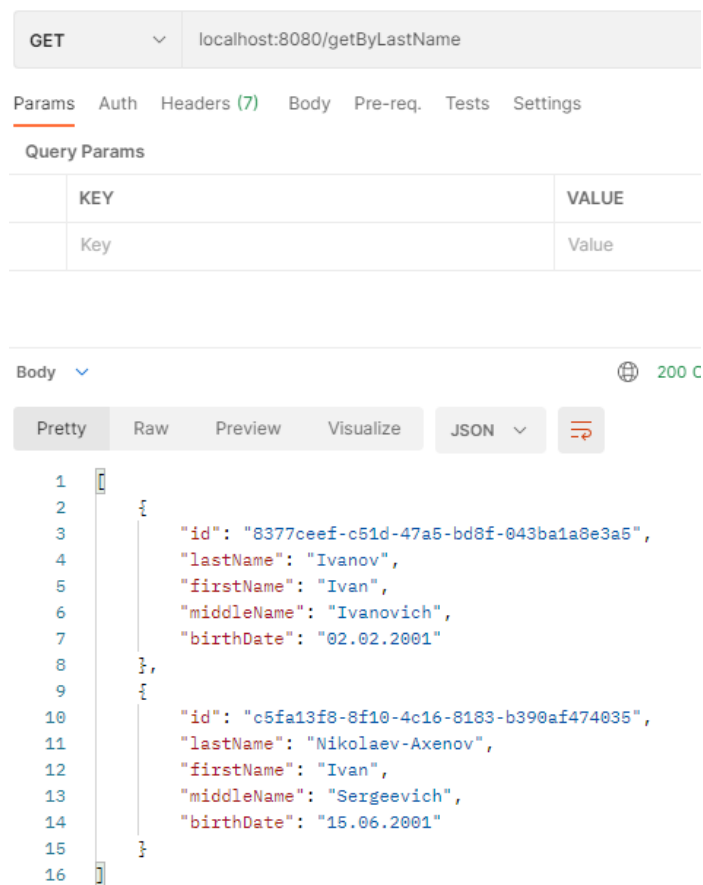


Рисунок 17.3 – Демонстрация работы программы

## Практическая работа №18

### *Цель работы*

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

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

### *Листинг программы*

#### *User.java*

Код не изменился с предыдущей задачи.

#### *Post.java*

Код не изменился с предыдущей задачи.

#### *UserRepository.java*

```
1. package app.Application.Interfaces;
2.
3. import app.Application.Classes.User;
4. import com.sun.istack.NotNull;
5. import org.springframework.data.jpa.repository.JpaRepository;
6. import org.springframework.stereotype.Repository;
7.
8. import java.util.List;
9. import java.util.UUID;
10.
11. @Repository("UserRepository")
12. public interface UserRepository extends JpaRepository<User, Long> {
13.     List<User> findAllByFirstName(String firstName);
14.     List<User> findAllByLastName(String lastName);
15.
16.     @NotNull List<User> findAll();
17.     void deleteById(UUID id);
18. }
19.
```

## *PostRepository.java*

```
1. package app.Application.Interfaces;
2.
3. import app.Application.Classes.Post;
4. import com.sun.istack.NotNull;
5. import org.springframework.data.jpa.repository.JpaRepository;
6. import org.springframework.stereotype.Repository;
7.
8. import java.util.List;
9. import java.util.UUID;
10.
11. @Repository("PostRepository")
12. public interface PostRepository extends JpaRepository<Post, Long> {
13.     Post findById(UUID id);
14.
15.     @NotNull List<Post> findAll();
16.     void deleteById(UUID id);
17. }
18.
```

## *UserService.java*

```
1. package app.Application.Services;
2.
3. import app.Application.Classes.User;
4. import app.Application.Interfaces.UserRepository;
5. import org.hibernate.Session;
6. import org.hibernate.SessionFactory;
7. import org.hibernate.query.Query;
8. import org.springframework.beans.factory.annotation.Autowired;
9. import org.springframework.stereotype.Service;
10.
11. import javax.annotation.PostConstruct;
12. import javax.annotation.PreDestroy;
13. import javax.persistence.criteria.CriteriaBuilder;
14. import javax.persistence.criteria.CriteriaQuery;
15. import javax.persistence.criteria.Root;
16. import java.util.List;
17. import java.util.UUID;
18.
19. @Service
20. public class UserService {
21.     @Autowired
22.     private final UserRepository userRepository;
23.
24.     public UserService(UserRepository userRepository) {
25.         this.userRepository = userRepository;
26.     }
27.
```



```

28.     public void addUser(User user) {
29.         userRepository.save(user);
30.     }
31.
32.     public List<User> getUsers() {
33.         return userRepository.findAll();
34.     }
35.
36.     public void deleteUser(UUID id) {
37.         userRepository.deleteById(id);
38.     }
39.
40.     public List<User> getByFirstName(String firstName) {
41.         return userRepository.findAllByFirstName(firstName);
42.     }
43.
44.     public List<User> getByLastName(String lastName) {
45.         return userRepository.findAllByLastName(lastName);
46.     }
47. }
48.

```

## *PostService.java*

```

1.  package app.Application.Services;
2.
3.  import app.Application.Classes.Post;
4.  import app.Application.Classes.User;
5.  import app.Application.Interfaces.PostRepository;
6.  import org.hibernate.Session;
7.  import org.hibernate.SessionFactory;
8.  import org.hibernate.query.Query;
9.  import org.springframework.beans.factory.annotation.Autowired;
10. import org.springframework.stereotype.Service;
11.
12. import javax.annotation.PostConstruct;
13. import javax.annotation.PreDestroy;
14. import javax.persistence.criteria.CriteriaBuilder;
15. import javax.persistence.criteria.CriteriaQuery;
16. import javax.persistence.criteria.Root;
17. import java.util.List;
18. import java.util.UUID;
19.
20. @Service
21. public class PostService {
22.     @Autowired
23.     private final PostRepository postRepository;
24.
25.     public PostService(PostRepository postRepository) {
26.         this.postRepository = postRepository;

```

```

27.     }
28.
29.     public void addPost(Post post) {
30.         postRepository.save(post);
31.     }
32.
33.     public List<Post> getPosts() {
34.         return postRepository.findAll();
35.     }
36.
37.     public void deletePost(UUID id) {
38.         postRepository.deleteById(id);
39.     }
40.
41.     public User getUserByPost(UUID id) {
42.         return postRepository.findById(id).getUser();
43.     }
44. }
45.

```

### *UserController.java*

Код не изменился с предыдущей задачи.

### *PostController.java*

```

1. package app.Application.Controllers;
2.
3. import app.Application.Classes.Post;
4. import app.Application.Classes.User;
5. import app.Application.Services.PostService;
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.web.bind.annotation.*;
8.
9. import java.util.List;
10. import java.util.UUID;
11.
12. @RestController
13. public class PostController {
14.     @Autowired
15.     private PostService postService;
16.
17.     @PostMapping("/posts")
18.     public void addPost(@RequestBody Post post) {
19.         postService.addPost(post);
20.     }
21.

```

```

22.     @GetMapping("/posts")
23.     public List<Post> getAll() {
24.         return postService.getPosts();
25.     }
26.
27.     @DeleteMapping("/post/{id}")
28.     public void delete(@PathVariable UUID id) {
29.         postService.deletePost(id);
30.     }
31.
32.     @GetMapping(value = "/post/{id}/user")
33.     public @ResponseBody
34.     User getUser(@PathVariable("id") UUID id) {
35.         return postService.getUserByPost(id);
36.     }
37. }
38.

```

## *Config.java*

```

1. package app.Application.Configuration;
2.
3. import org.springframework.context.annotation.Configuration;
4. import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
5.
6. @Configuration
7. @EnableJpaRepositories(basePackages = {"app.Application"})
8. public class Config {
9. }
10.

```

*application.yml* (в следующих работах тоже присутствует, но меняется только ссылка на базу данных)

```

1. spring:
2.   jpa:
3.     database: POSTGRESQL
4.     show-sql: true
5.     hibernate:
6.       ddl-auto: create-drop
7.   datasource:
8.     platform: postgres
9.     url: jdbc:postgresql://localhost:5432/pr18db
10.    username: postgres
11.    password: secret
12.    driverClassName: org.postgresql.Driver

```

## Результат выполнения программы

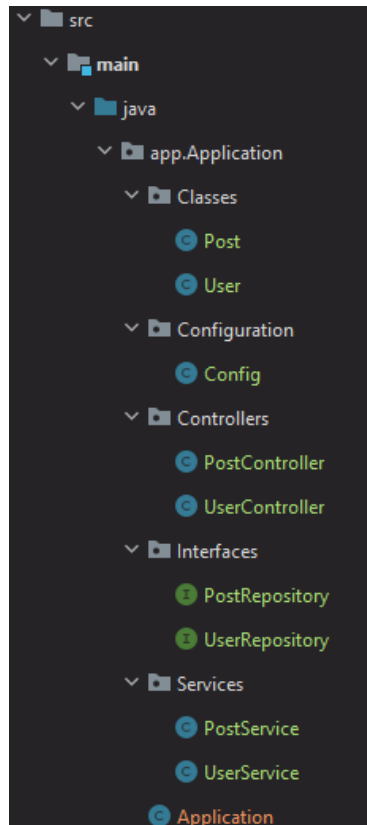


Рисунок 18.1 – Демонстрация работы программы

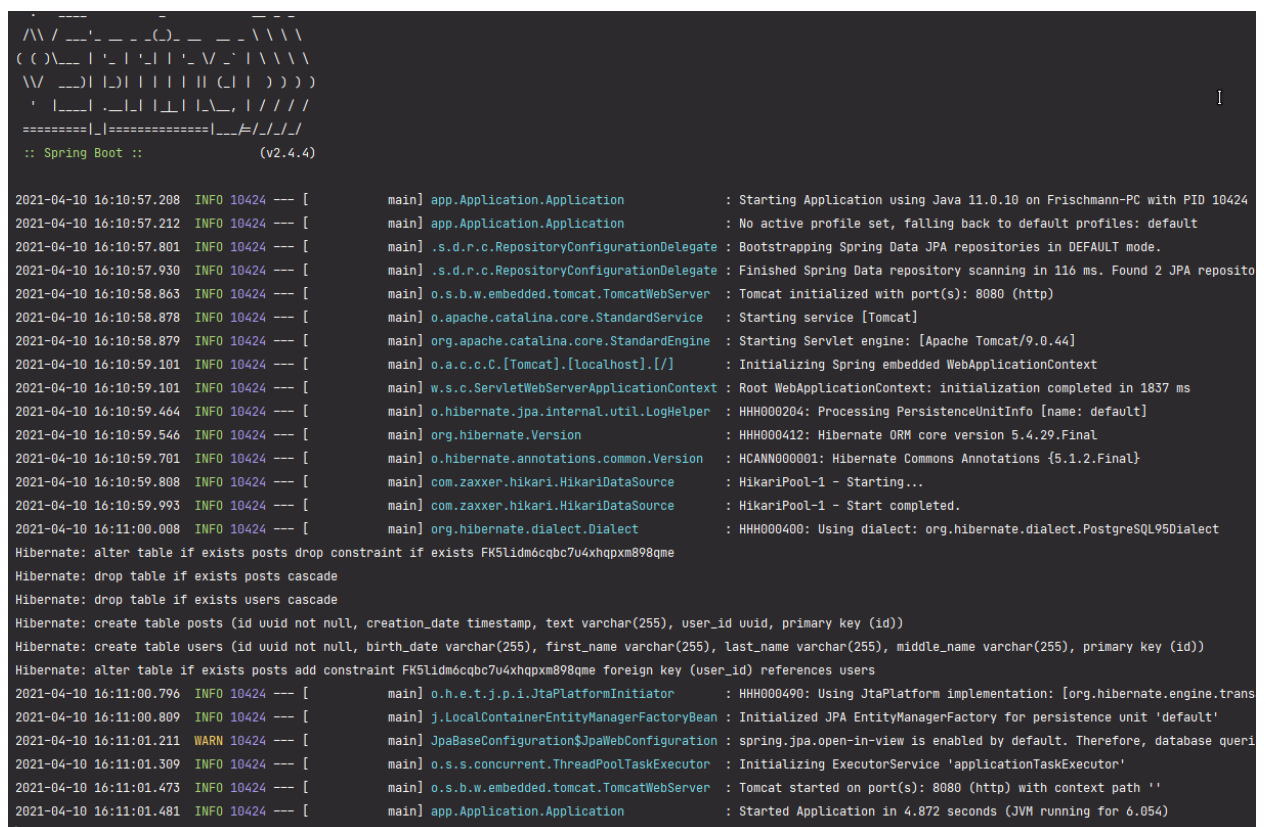


Рисунок 18.2 – Демонстрация работы программы

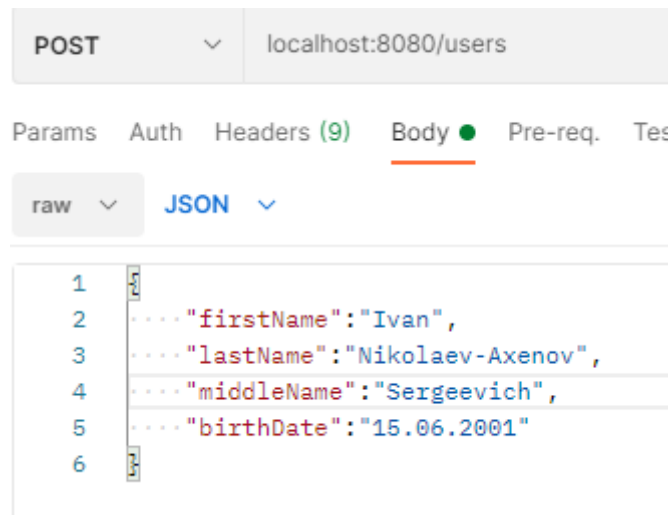


Рисунок 18.3 – Демонстрация работы программы

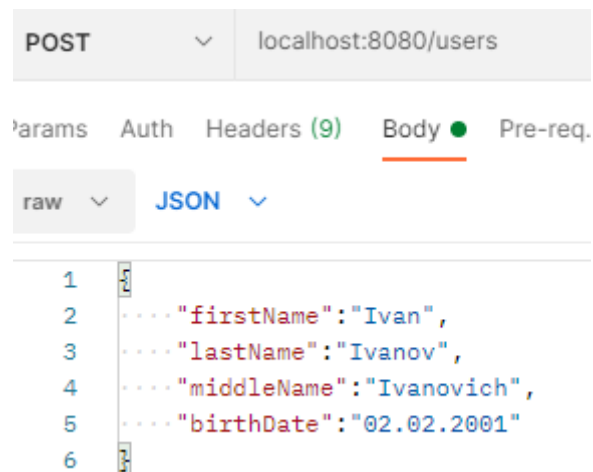


Рисунок 18.4 – Демонстрация работы программы

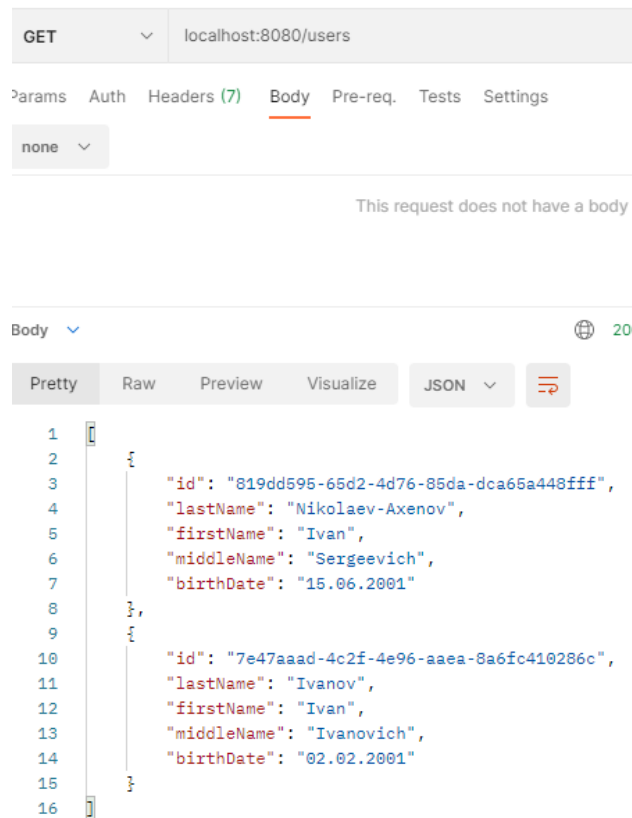


Рисунок 18.5 – Демонстрация работы программы

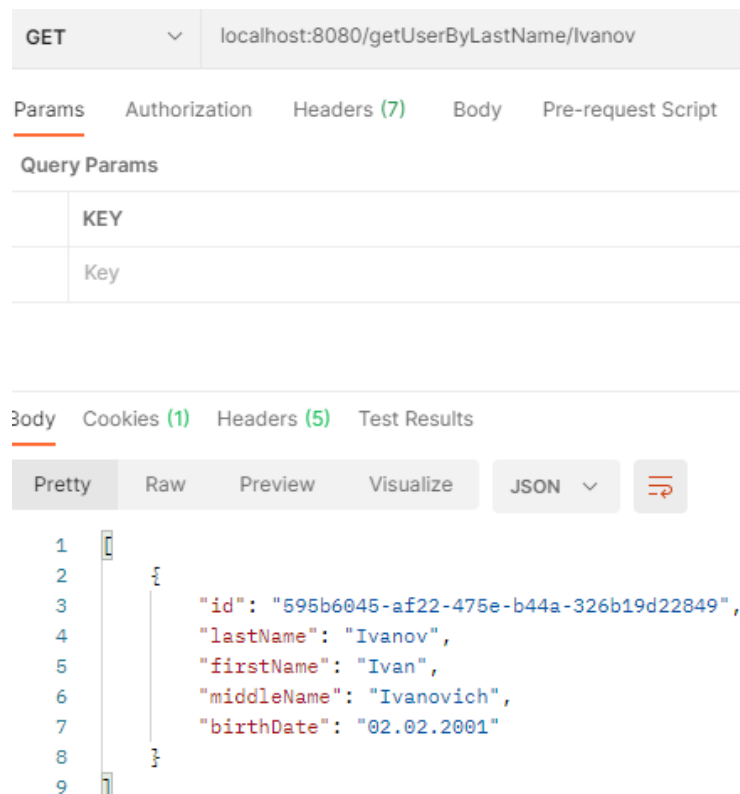


Рисунок 18.6 – Демонстрация работы программы

## Практическая работа №19

### ***Цель работы***

Тема: Знакомство с логированием с использованием Logback в Spring.

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

### ***Листинг программы***

#### ***User.java***

Код не изменился с предыдущей задачи.

#### ***Post.java***

Код не изменился с предыдущей задачи.

#### ***UserRepository.java***

Код не изменился с предыдущей задачи.

#### ***PostRepository.java***

Код не изменился с предыдущей задачи.

#### ***UserService.java***

Код не изменился с предыдущей задачи.

#### ***PostService.java***

Код не изменился с предыдущей задачи.

#### ***UserController.java***

Код не изменился с предыдущей задачи.





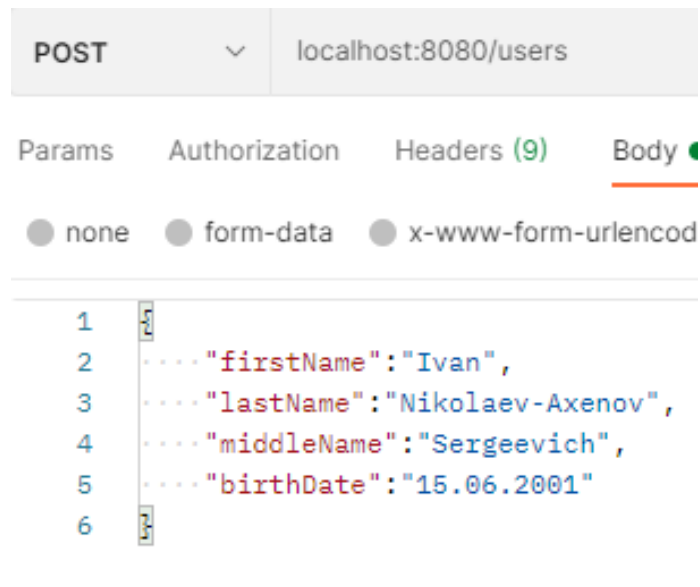


Рисунок 19.2 – Демонстрация работы программы

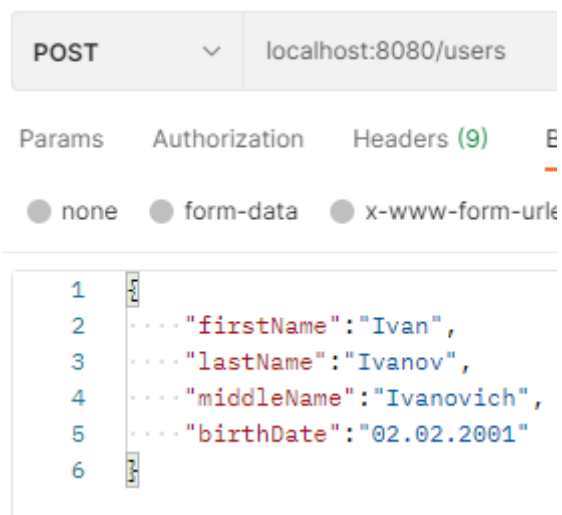


Рисунок 19.3 – Демонстрация работы программы

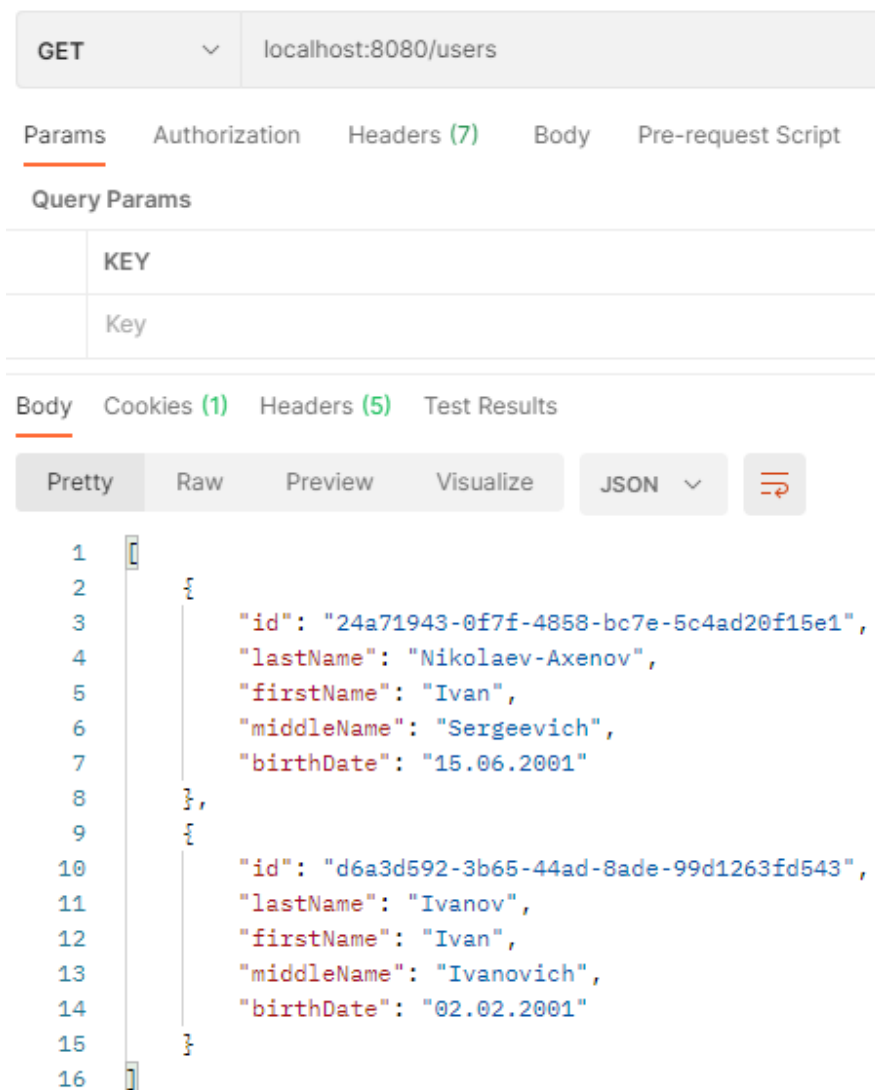


Рисунок 19.4 – Демонстрация работы программы

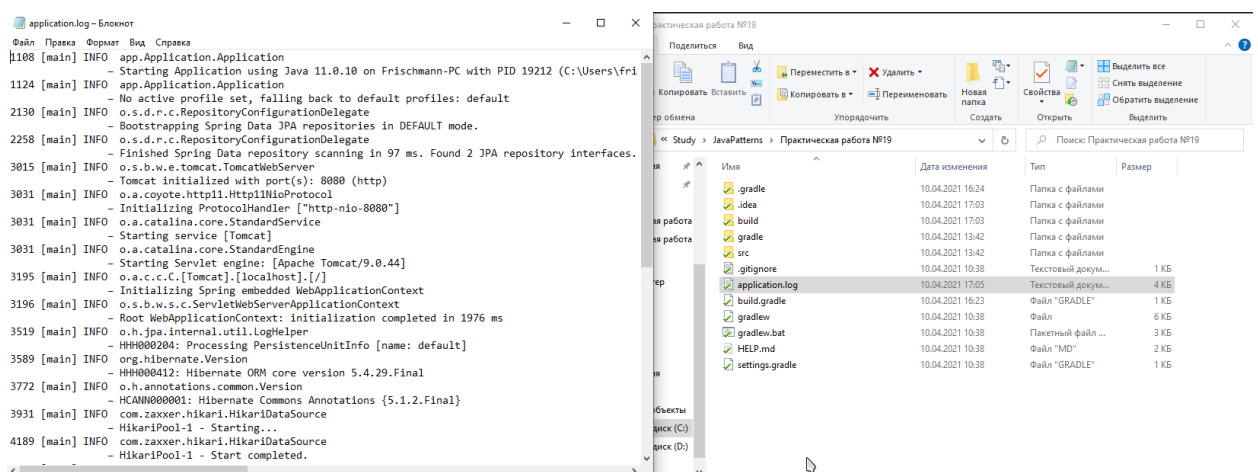


Рисунок 19.5 – Демонстрация работы программы

## Практическая работа №20

### ***Цель работы***

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

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

### ***Листинг программы***

#### ***User.java***

Код не изменился с предыдущей задачи.

#### ***Post.java***

Код не изменился с предыдущей задачи.

#### ***UserRepository.java***

Код не изменился с предыдущей задачи.

#### ***PostRepository.java***

Код не изменился с предыдущей задачи.

#### ***UserService.java***

Код не изменился с предыдущей задачи.

#### ***PostService.java***

Код не изменился с предыдущей задачи.

#### ***UserController.java***

Код не изменился с предыдущей задачи.

## *PostController.java*

Код не изменился с предыдущей задачи.

## *Config.java*

Код не изменился с предыдущей задачи.

## *Aspect.java*

```
1. package app.Application;
2.
3. import lombok.extern.slf4j.Slf4j;
4. import org.aspectj.lang.ProceedingJoinPoint;
5. import org.aspectj.lang.annotation.Around;
6. import org.aspectj.lang.annotation.Pointcut;
7. import org.springframework.stereotype.Component;
8.
9. import java.util.logging.Logger;
10.
11. @Slf4j
12. @Component
13. @org.aspectj.lang.annotation.Aspect
14. public class Aspect {
15.     private Logger log = Logger.getLogger(Aspect.class.getName());
16.
17.     @Around("allServiceMethods()")
18.     public Object logExecutionTime (ProceedingJoinPoint joinPoint) throws Throwable
19.     {
20.         long start = System.currentTimeMillis();
21.         Object proceed = joinPoint.proceed();
22.         long executionTime = System.currentTimeMillis() - start;
23.         log.info(joinPoint.getSignature() + " выполнен за " + executionTime +
24.             "мс");
25.         return proceed;
26.     }
27.
28.     @Pointcut("within(app.Application.Services.*)")
29.     public void allServiceMethods() {}
30. }
```

## Результат выполнения программы

```

  ____  __
 / ___/  / /
/ /   /  / /
/ /___/  / /
/_____/  / /
        / /
       /_/

( ( ) \___ | _ | ' _ | ' _ \ ___ | \ \ \ \
 \ \ ___ | _ | | | | | | | | | | | | | | |
  ' | ___ | _ | | | | | | | | | | | | | | |
=====|_|=====|_|_#/_/_/_/

:: Spring Boot ::                (v2.4.4)

17:21:00.548 [main] INFO
    app.Application.Application - Starting Application using Java 11.0.10 on Frischmann-PC with PID 6064 (C:\Users\frisc\GoogleDrive\St
17:21:00.551 [main] INFO
    app.Application.Application - No active profile set, falling back to default profiles: default
17:21:01.174 [main] INFO
    o.s.d.r.c.RepositoryConfigurationDelegate - Bootstrapping Spring Data JPA repositories in DEFAULT mode.
17:21:01.252 [main] INFO
    o.s.d.r.c.RepositoryConfigurationDelegate - Finished Spring Data repository scanning in 65 ms. Found 2 JPA repository interfaces.
17:21:02.514 [main] INFO
    o.s.b.w.e.tomcat.TomcatWebServer - Tomcat initialized with port(s): 8080 (http)
17:21:02.524 [main] INFO
    o.a.coyote.http11.Http11NioProtocol - Initializing ProtocolHandler ["http-nio-8080"]
17:21:02.525 [main] INFO
    o.a.catalina.core.StandardService - Starting service [Tomcat]
17:21:02.525 [main] INFO
    o.a.catalina.core.StandardEngine - Starting Servlet engine: [Apache Tomcat/9.0.44]
17:21:02.650 [main] INFO
    o.a.c.c.C.[Tomcat].[localhost].[/] - Initializing Spring embedded WebApplicationContext
17:21:02.651 [main] INFO
    o.s.b.w.s.c.ServletWebServerApplicationContext - Root WebApplicationContext: initialization completed in 2042 ms
17:21:02.896 [main] INFO
    o.h.jpa.internal.util.LogHelper - HHH000204: Processing PersistenceUnitInfo [name: default]
17:21:02.965 [main] INFO
    org.hibernate.Version - HHH000412: Hibernate ORM core version 5.4.29.Final
17:21:03.205 [main] INFO
    o.h.annotations.common.Version - HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
17:21:03.467 [main] INFO
    com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
17:21:03.695 [main] INFO

```

### Рисунок 20.1 – Демонстрация работы программы

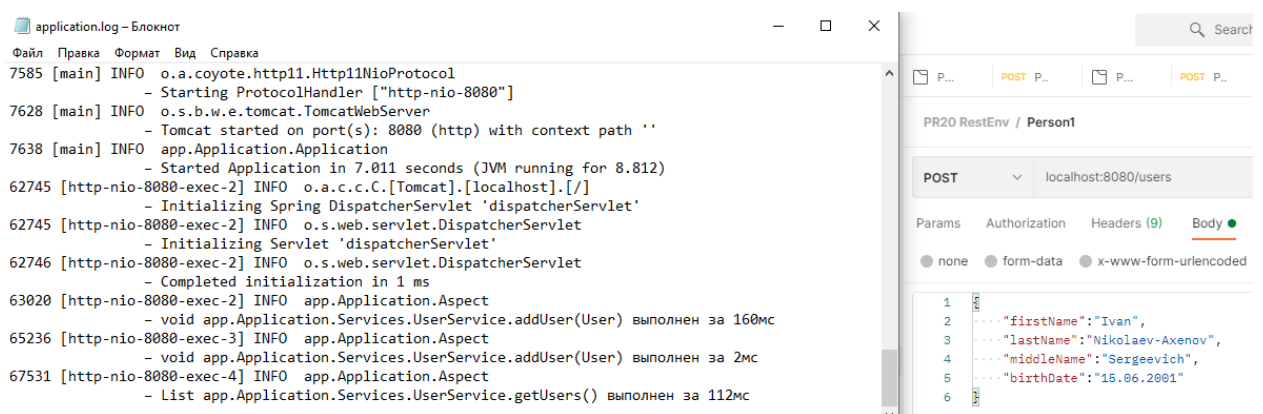


Рисунок 20.2 – Демонстрация работы программы

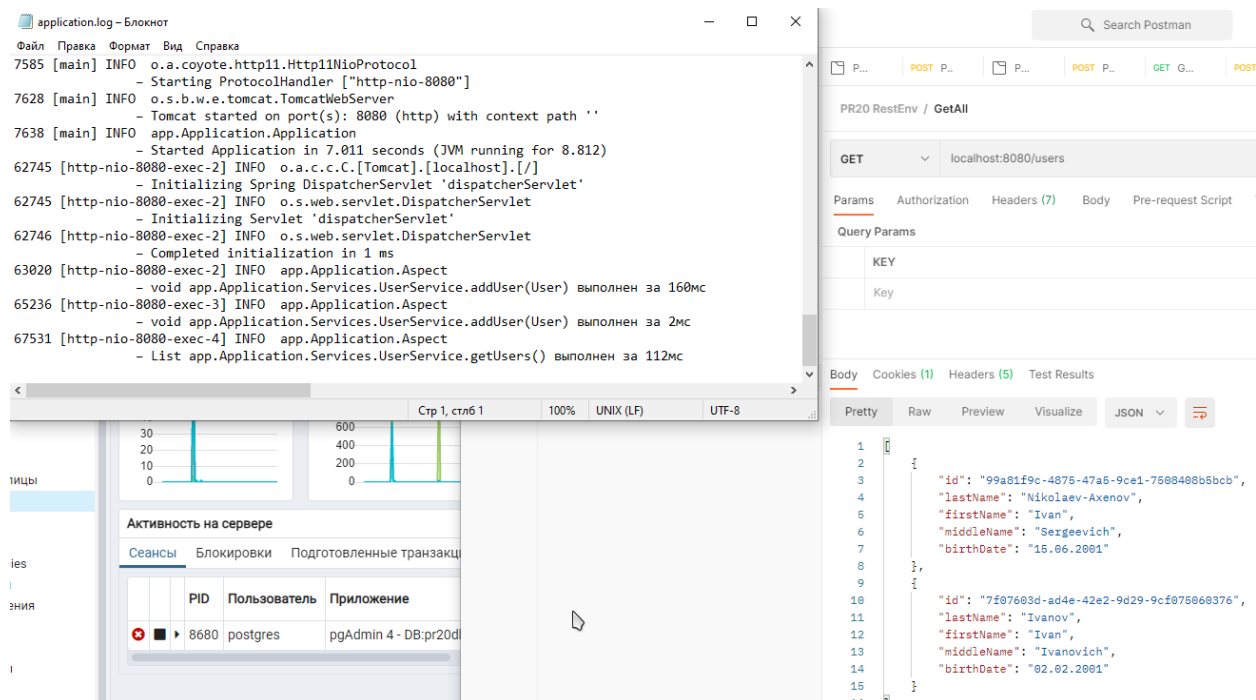


Рисунок 20.3 – Демонстрация работы программы

## Практическая работа №21

### ***Цель работы***

Тема: Проксирование. Аннотация Transactional. Аннотация Async.

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

### ***Листинг программы***

#### ***User.java***

Код не изменился с предыдущей задачи.

#### ***Post.java***

Код не изменился с предыдущей задачи.

#### ***UserRepository.java***

Код не изменился с предыдущей задачи.

#### ***PostRepository.java***

Код не изменился с предыдущей задачи.

#### ***UserService.java***

Код не изменился с предыдущей задачи.

#### ***PostService.java***

Код не изменился с предыдущей задачи.

## *UserController.java*

Код не изменился с предыдущей задачи.

## *PostController.java*

Код не изменился с предыдущей задачи.

## *Config.java*

```
1. package app.Application.Configuration;
2.
3. import app.Application.EmailService;
4. import org.springframework.context.annotation.Bean;
5. import org.springframework.context.annotation.Configuration;
6. import org.springframework.context.annotation.EnableAspectJAutoProxy;
7. import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
8. import org.springframework.mail.javamail.JavaMailSender;
9. import org.springframework.mail.javamail.JavaMailSenderImpl;
10. import org.springframework.scheduling.annotation.EnableAsync;
11.
12. import java.util.Properties;
13.
14. @Configuration
15. @EnableAspectJAutoProxy
16. @EnableJpaRepositories(basePackages = {"app.Application"})
17. @EnableAsync
18. public class Config {
19.     @Bean
20.     public JavaMailSender getJavaMailSender() {
21.         JavaMailSenderImpl mailSender = new JavaMailSenderImpl();
22.         mailSender.setHost("smtp.mail.ru");
23.         mailSender.setPort(465);
24.
25.         mailSender.setUsername("lorememail@bk.ru");
26.         mailSender.setPassword("secret");
27.
28.         Properties props = mailSender.getJavaMailProperties();
29.         props.put("mail.transport.protocol", "smtps");
30.         props.put("mail.smtp.auth", "true");
31.         props.put("smtp.ssl.enable", "true");
32.         props.put("mail.smtp.starttls.enable", "true");
33.         props.put("mail.debug", "true");
34.
35.         return mailSender;
36.     }
37.     @Bean
```



```

38.     public EmailService getEmailService(){
39.         return new EmailService();
40.     }
41. }
42.

```

## *Aspect.java*

Код не изменился с предыдущей задачи.

## *EmailService.java*

```

1. package app.Application;
2.
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.mail.SimpleMailMessage;
5. import org.springframework.mail.javamail.JavaMailSender;
6. import org.springframework.scheduling.annotation.Async;
7.
8. public class EmailService {
9.     @Autowired
10.    public JavaMailSender emailSender;
11.
12.    @Async
13.    public void SendEmail(){
14.        SimpleMailMessage message = new SimpleMailMessage();
15.
16.        message.setFrom("lorememail@bk.ru");
17.        message.setTo("ghost777t@ya.ru");
18.        message.setSubject("Test email message");
19.        message.setText("Lorem ipsum dolor sit amet, consectetur adipiscing elit.
        Nulla feugiat eget sapien sed lacinia.");
20.
21.        this.emailSender.send(message);
22.        System.out.println("Email successfully sent!");
23.    }
24. }
25.

```

## Результат выполнения программы

```

/\ / _\ ' _ _ _ \ ) _ _ _ \\ \ \ \
( ( ) \_ _ _ | _ _ | ' _ | ' _ V _ ' | \ \ \ \
\\ / _ _ ) | _ ) | | | | | | ( _ | ) ) ) )
' | _ _ _ | _ _ | | _ | | _ \ , | / / / /
=====|_|=====|___#/_/_/_/

:: Spring Boot ::                (v2.4.4)

23:21:25.220 [main] INFO
    app.Application.Application - Starting Application using Java 11.0.10 on Frischmann-PC with PID 5916 (C:\Users\frisc\GoogleDrive\Study\JavaPat
23:21:25.222 [main] INFO
    app.Application.Application - No active profile set, falling back to default profiles: default
23:21:25.806 [main] INFO
    o.s.d.r.c.RepositoryConfigurationDelegate - Bootstrapping Spring Data JPA repositories in DEFAULT mode.
23:21:25.886 [main] INFO
    o.s.d.r.c.RepositoryConfigurationDelegate - Finished Spring Data repository scanning in 67 ms. Found 2 JPA repository interfaces.
23:21:26.787 [main] INFO
    o.s.b.w.e.tomcat.TomcatWebServer - Tomcat initialized with port(s): 8080 (http)
23:21:26.799 [main] INFO
    o.a.coyote.http11.Http11NioProtocol - Initializing ProtocolHandler ["http-nio-8080"]
23:21:26.800 [main] INFO
    o.a.catalina.core.StandardService - Starting service [Tomcat]
23:21:26.800 [main] INFO
    o.a.catalina.core.StandardEngine - Starting Servlet engine: [Apache Tomcat/9.0.44]
23:21:26.922 [main] INFO
    o.a.c.c.C.[Tomcat].[localhost].[/] - Initializing Spring embedded WebApplicationContext
23:21:26.922 [main] INFO
    o.s.b.w.s.c.ServletWebServerApplicationContext - Root WebApplicationContext: initialization completed in 1655 ms
23:21:27.140 [main] INFO
    o.h.jpa.internal.util.LogHelper - HHH0000204: Processing PersistenceUnitInfo [name: default]
23:21:27.275 [main] INFO
    org.hibernate.Version - HHH0000412: Hibernate ORM core version 5.4.29.Final
23:21:27.460 [main] INFO
    o.h.annotations.common.Version - HCANN000001: Hibernate Commons Annotations {5.1.2.Final}

```

### Рисунок 21.1 – Демонстрация работы программы

POST

localhost:8080/users

Params

Authorization

Headers (8)

Body

☐ none

☐ form-data

☐ x-www-form-urlencoded

```
1 {  
2   ... "firstName": "Ivan",  
3   ... "lastName": "Nikolaev-Axenov",  
4   ... "middleName": "Sergeevich",  
5   ... "birthDate": "15.06.2001"  
6 }
```

Рисунок 21.2 – Демонстрация работы программы

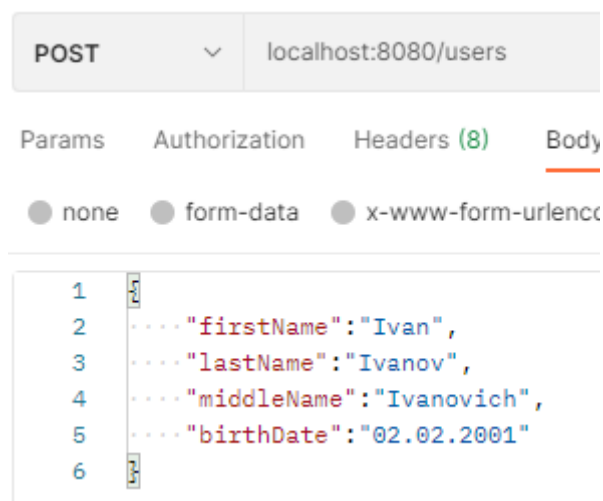


Рисунок 21.3 – Демонстрация работы программы

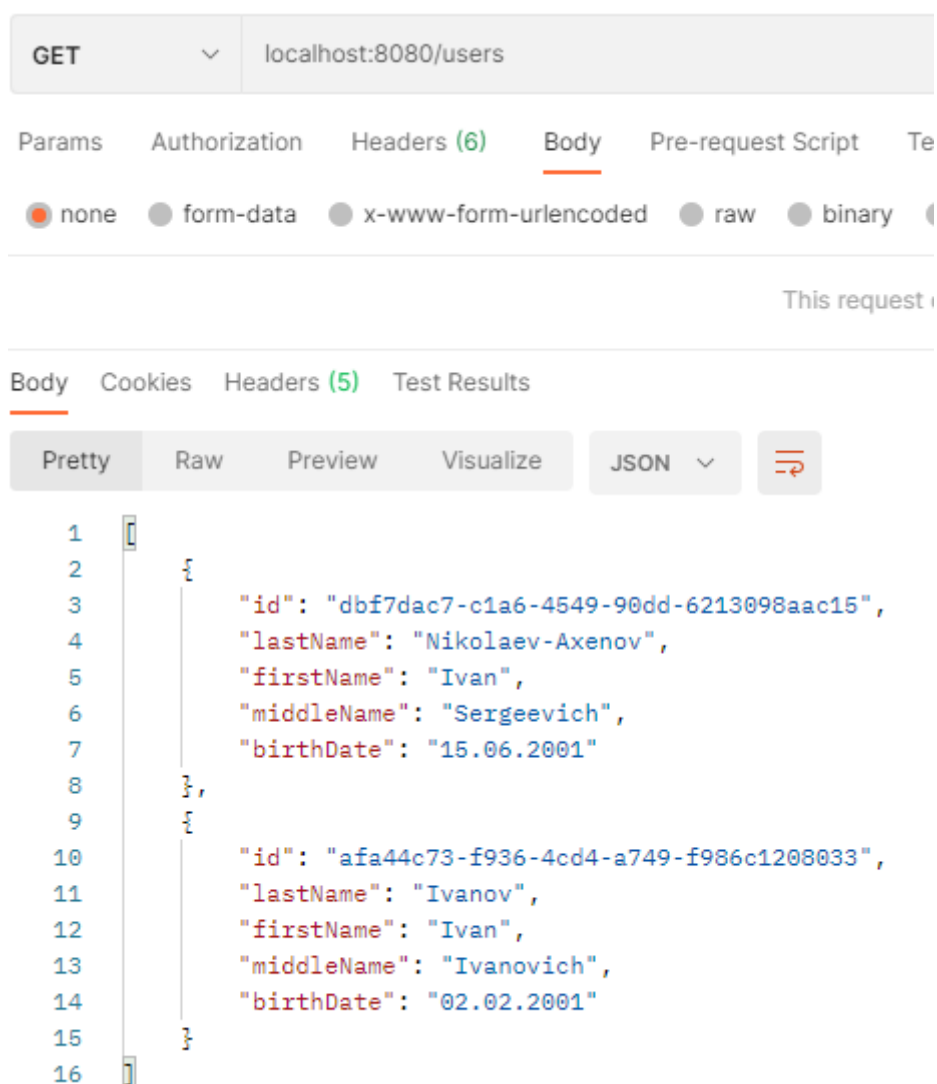


Рисунок 21.4 – Демонстрация работы программы

```

DEBUG SMTP: Found extension "PIPELINING", arg
DEBUG SMTP: Found extension "AUTH", arg "PLAIN LOGIN XOAUTH2"
DEBUG SMTP: protocolConnect login, host=smtp.mail.ru, user=lorememail@bk.ru, password=<non-null>
DEBUG SMTP: Attempt to authenticate using mechanisms: LOGIN PLAIN DIGEST-MD5 NTLM XOAUTH2
DEBUG SMTP: Using mechanism LOGIN
DEBUG SMTP: AUTH LOGIN command trace suppressed
DEBUG SMTP: AUTH LOGIN succeeded
DEBUG SMTP: use8bit false
MAIL FROM:<lorememail@bk.ru>
250 OK
RCPT TO:<ghost777t@ya.ru>
250 Accepted
DEBUG SMTP: Verified Addresses
DEBUG SMTP:  ghost777t@ya.ru
DATA
354 Enter message, ending with "." on a line by itself
Date: Sat, 10 Apr 2021 23:35:43 +0300 (MSK)
From: lorememail@bk.ru
To: ghost777t@ya.ru
Message-ID: <1849625555.0.1618086943511@Frischmann-PC>
Subject: Test email message
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla feugiat eget sapien sed lacinia.
.
250 OK id=1LVKKV-0008Pz-Bw
DEBUG SMTP: message successfully delivered to mail server
QUIT
221 smtp57.i.mail.ru closing connection
Email successfully sent!
Hibernate: select user0_.id as id1_1_, user0_.birth_date as birth_da2_1_, user0_.first_name as first_na3_1_, user0_.last_name as last_na4_1_ from user0_ where user0_.id=1
23:35:43.956 [http-nio-8080-exec-3] INFO
app.Application.Aspect - List app.Application.Services.UserService.getUsers() выполнен за 111мс

```

Рисунок 21.5 – Демонстрация работы программы

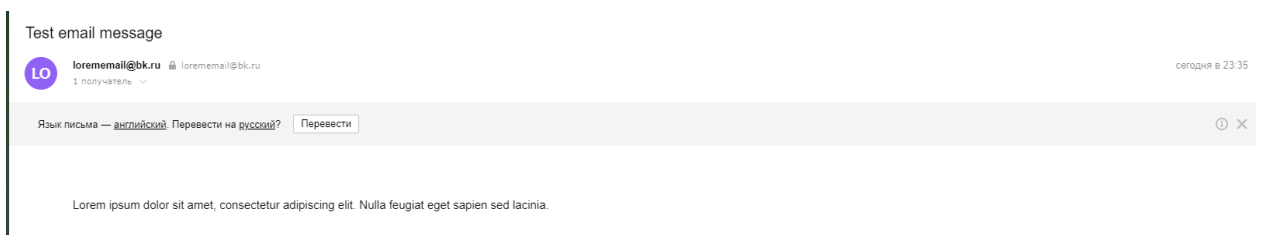


Рисунок 21.6 – Демонстрация работы программы

## Практическая работа №22

### ***Цель работы***

Тема: Планирование заданий. Scheduler в Spring.

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

### ***Листинг программы***

#### ***User.java***

Код не изменился с предыдущей задачи.

#### ***Post.java***

Код не изменился с предыдущей задачи.

#### ***UserRepository.java***

Код не изменился с предыдущей задачи.

#### ***PostRepository.java***

Код не изменился с предыдущей задачи.

#### ***UserService.java***

Код не изменился с предыдущей задачи.

#### ***PostService.java***

Код не изменился с предыдущей задачи.

#### ***UserController.java***

Код не изменился с предыдущей задачи.

## *PostController.java*

Код не изменился с предыдущей задачи.

## *Config.java*

```
1. package app.Application.Configuration;
2.
3. import app.Application.Services.EmailService;
4. import app.Application.Services.Schedule;
5. import org.springframework.context.annotation.Bean;
6. import org.springframework.context.annotation.Configuration;
7. import org.springframework.context.annotation.EnableAspectJAutoProxy;
8. import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
9. import org.springframework.mail.javamail.JavaMailSender;
10. import org.springframework.mail.javamail.JavaMailSenderImpl;
11. import org.springframework.scheduling.annotation.EnableAsync;
12. import org.springframework.scheduling.annotation.EnableScheduling;
13.
14. import java.util.Properties;
15.
16. @Configuration
17. @EnableAspectJAutoProxy
18. @EnableJpaRepositories(basePackages = {"app.Application"})
19. @EnableAsync
20. @EnableScheduling
21. public class Config {
22.     @Bean
23.     public JavaMailSender getJavaMailSender() {
24.         JavaMailSenderImpl mailSender = new JavaMailSenderImpl();
25.         mailSender.setHost("smtp.mail.ru");
26.         mailSender.setPort(465);
27.
28.         mailSender.setUsername("lorememail@bk.ru");
29.         mailSender.setPassword("secret");
30.
31.         Properties props = mailSender.getJavaMailProperties();
32.         props.put("mail.transport.protocol", "smtps");
33.         props.put("mail.smtp.auth", "true");
34.         props.put("smtp.ssl.enable", "true");
35.         props.put("mail.smtp.starttls.enable", "true");
36.         props.put("mail.debug", "true");
37.
38.         return mailSender;
39.     }
40.
41.     @Bean
```

```

42.     public EmailService getEmailService() {
43.         return new EmailService();
44.     }
45. }
46.

```

### *Aspect.java*

Код не изменился с предыдущей задачи.

### *EmailService.java*

Код не изменился с предыдущей задачи.

### *ScheduleMXBean.java*

```

1. package app.Application;
2.
3. import java.io.IOException;
4. import app.Application.Interfaces.UserRepository;
5. import app.Application.Interfaces.PostRepository;
6.
7. public interface ScheduleMXBean {
8.     void doScheduledTask() throws IOException;
9. }
10.

```

### *Schedule.java*

```

1. package app.Application.Services;
2.
3. import app.Application.Classes.Post;
4. import app.Application.Classes.User;
5. import app.Application.Controllers.PostController;
6. import app.Application.Controllers.UserController;
7. import app.Application.Interfaces.PostRepository;
8. import app.Application.Interfaces.UserRepository;
9. import app.Application.ScheduleMXBean;
10. import org.springframework.beans.factory.annotation.Autowired;
11. import org.springframework.jmx.export.annotation.ManagedOperation;
12. import org.springframework.scheduling.annotation.Scheduled;
13. import org.springframework.stereotype.Service;
14.
15. import java.io.File;
16. import java.io.FileWriter;

```

```

17. import java.io.IOException;
18. import java.util.List;
19. import java.util.Objects;
20.
21. @Service
22. public class Schedule implements ScheduleMXBean {
23.     @Autowired
24.     private final UserRepository userRepository;
25.
26.     @Autowired
27.     private final PostRepository postRepository;
28.
29.     public Schedule(UserRepository userRepository, PostRepository postRepository) {
30.         this.userRepository = userRepository;
31.         this.postRepository = postRepository;
32.     }
33.
34.     private Boolean isEmpty(final File file) {
35.         return (file.isDirectory() && (file.list().length > 0));
36.     }
37.
38.     @ManagedOperation
39.     @Scheduled(cron = "0 0/2 * * * *")
40.     public void doScheduledTask() throws IOException {
41.         if (isEmpty(new
42.             File("C:\\Users\\frisc\\GoogleDrive\\Study\\JavaPatterns\\Практическая работа
43.             №22\\testDirectory"))) {
44.             for (File myFile : new
45.                 File("C:\\Users\\frisc\\GoogleDrive\\Study\\JavaPatterns\\Практическая работа
46.                 №22\\testDirectory").listFiles()) {
47.                 if (myFile.isFile()) myFile.delete();
48.             }
49.         }
50.
51.         List <Post> posts = postRepository.findAll();
52.         List <User> users = userRepository.findAll();
53.
54.         for (int i = 0; i < users.size(); i++) {
55.             File user = new
56.                 File("C:\\Users\\frisc\\GoogleDrive\\Study\\JavaPatterns\\Практическая работа
57.                 №22\\testDirectory\\user_" + i + ".txt");
58.             FileWriter writer = new FileWriter(user, true);
59.             System.out.println(users.get(i).toString());
60.             writer.write(users.get(i).toString());
61.             writer.close();
62.         }
63.
64.         for (int i = 0; i < posts.size(); i++) {
65.             File post = new
66.                 File("C:\\Users\\frisc\\GoogleDrive\\Study\\JavaPatterns\\Практическая работа
67.                 №22\\testDirectory\\post_" + i + ".txt");

```



```
59.         FileWriter writer = new FileWriter(post, true);
60.         writer.write(posts.get(i).toString());
61.         writer.close();
62.     }
63. }
64. }
```

### Результат выполнения программы

[illegible]

Рисунок 22.1 – Демонстрация работы программы

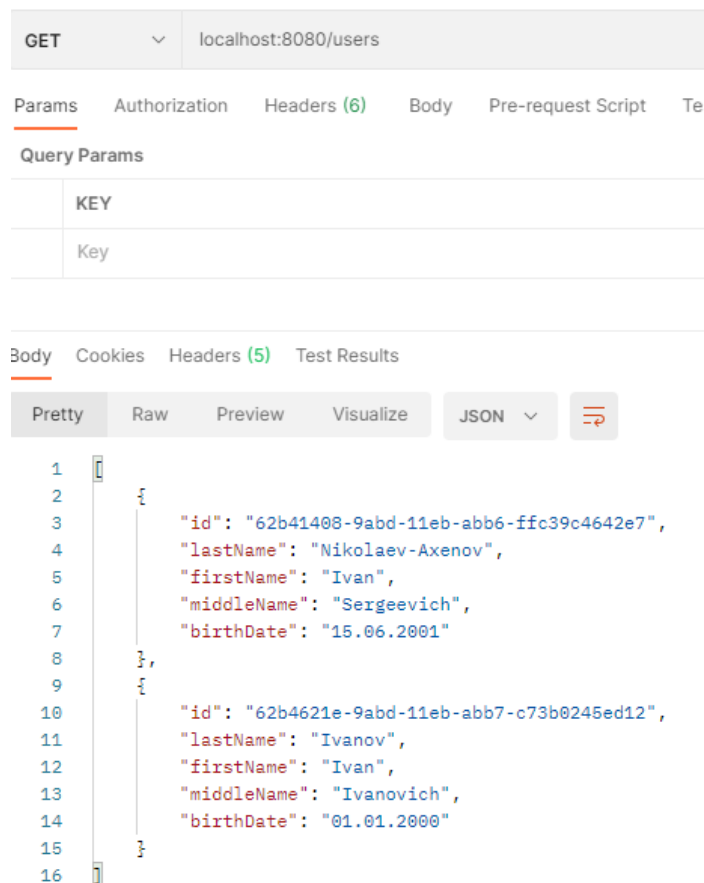


Рисунок 22.2 – Демонстрация работы программы

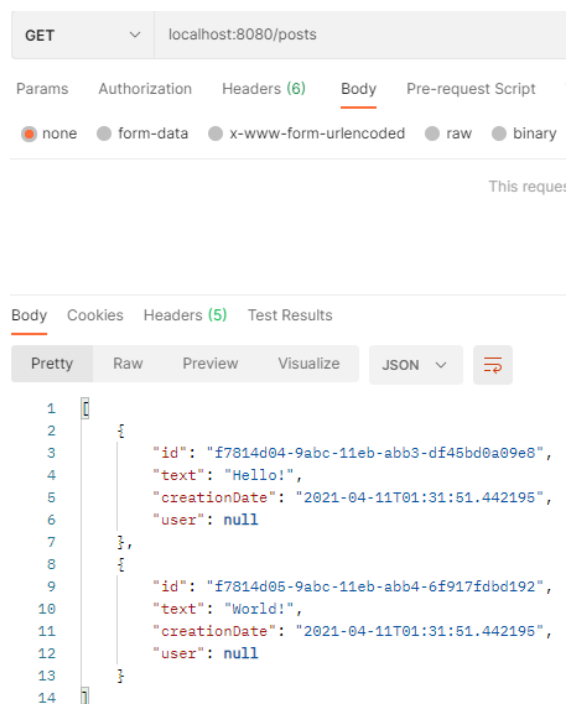


Рисунок 22.3 – Демонстрация работы программы

```

Hibernate: select user0_.id as id1_1_, user0_.birth_date as birth_da2_1_, user0_.first_name as first_na3_1_, user0_.last_name as last_nam4_1_, user0_.middle_name as middle_n5_1_ from users user0_
14:59:43.869 [http-nio-8080-exec-1] INFO
app.Application.Aspect - List app.Application.Services.UserService getUsers() выполнен за 142мс
Hibernate: select post0_.id as id1_0_, post0_.creation_date as creation2_0_, post0_.text as text3_0_, post0_.user_id as user_id4_0_ from posts post0_
Hibernate: select user0_.id as id1_1_, user0_.birth_date as birth_da2_1_, user0_.first_name as first_na3_1_, user0_.last_name as last_nam4_1_, user0_.middle_name as middle_n5_1_ from users user0_
User{id=62b41408-9abd-11eb-abb6-ffc39c4642e7, lastName='Nikolaev-Axenov', firstName='Ivan', middleName='Sergeevich', birthDate='15.06.2001'}
User{id=62b4621e-9abd-11eb-abb7-c73b0245ed12, lastName='Ivanov', firstName='Ivan', middleName='Ivanovich', birthDate='01.01.2000'}
15:00:00.035 [scheduling-1] INFO
app.Application.Aspect - void app.Application.Services.Schedule.doScheduledTask() выполнен за 31мс

```

Рисунок 22.4 – Демонстрация работы программы

JavaPatterns > Практическая работа №22 > testDirectory

Поиск: testDirectory

Имя

Дата изменения

Тип

Размер

post\_0.txt

11.04.2021 15:00

Файл "TXT"

1 КБ

post\_1.txt

11.04.2021 15:00

Файл "TXT"

1 КБ

user\_0.txt

11.04.2021 15:00

Файл "TXT"

1 КБ

user\_1.txt

11.04.2021 15:00

Файл "TXT"

1 КБ

Рисунок 22.5 – Демонстрация работы программы

```

user_0.txt
1 |User{id=62b41408-9abd-11eb-abb6-ffc39c4642e7, lastName='Nikolaev-Axenov', firstName='Ivan', middleName='Sergeevich', birthDate='15.06.2001'}

```

Рисунок 22.6 – Демонстрация работы программы

## Практическая работа №23

### *Цель работы*

Тема: Использование Spring Security для аутентификации и авторизации пользователей.

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

### *Листинг программы*

#### *User.java*

```
1. package app.Application.Classes;
2.
3. import org.springframework.security.core.GrantedAuthority;
4. import org.springframework.security.core.userdetails.UserDetails;
5.
6. import javax.persistence.*;
7. import javax.validation.constraints.Size;
8. import java.util.Collection;
9. import java.util.Set;
10.
11. @Entity
12. @Table(name = "t_user")
13. public class User implements UserDetails {
14.     @Id
15.     @GeneratedValue(strategy = GenerationType.IDENTITY)
16.     private Long id;
17.
18.     @Size(min=2, message = "Не меньше 5 знаков")
19.     private String username;
20.
21.     @Size(min=2, message = "Не меньше 5 знаков")
22.     private String password;
23.
24.     @Transient
25.     private String passwordConfirm;
26.
27.     @ManyToMany(fetch = FetchType.EAGER)
28.     private Set<Role> roles;
29.
30.     public User() {
31.     }
32.
33.     public Long getId() {
```

```

34.         return id;
35.     }
36.
37.     public void setId(Long id) {
38.         this.id = id;
39.     }
40.
41.     @Override
42.     public String getUsername() {
43.         return username;
44.     }
45.
46.     @Override
47.     public boolean isAccountNonExpired() {
48.         return true;
49.     }
50.
51.     @Override
52.     public boolean isAccountNonLocked() {
53.         return true;
54.     }
55.
56.     @Override
57.     public boolean isCredentialsNonExpired() {
58.         return true;
59.     }
60.
61.     @Override
62.     public boolean isEnabled() {
63.         return true;
64.     }
65.
66.     public void setUsername(String username) {
67.         this.username = username;
68.     }
69.
70.     @Override
71.     public Collection<? extends GrantedAuthority> getAuthorities() {
72.         return getRoles();
73.     }
74.
75.     @Override
76.     public String getPassword() {
77.         return password;
78.     }
79.
80.     public void setPassword(String password) {
81.         this.password = password;
82.     }
83.

```

```

84.     public String getPasswordConfirm() {
85.         return passwordConfirm;
86.     }
87.
88.     public void setPasswordConfirm(String passwordConfirm) {
89.         this.passwordConfirm = passwordConfirm;
90.     }
91.
92.     public Set<Role> getRoles() {
93.         return roles;
94.     }
95.
96.     public void setRoles(Set<Role> roles) {
97.         this.roles = roles;
98.     }
99.
100.    }
101.

```

## *Role.java*

```

1.  package app.Application.Classes;
2.
3.  import org.springframework.security.core.GrantedAuthority;
4.
5.  import javax.persistence.*;
6.  import java.util.Set;
7.
8.  @Entity
9.  @Table(name = "t_role")
10. public class Role implements GrantedAuthority {
11.     @Id
12.     private Long id;
13.
14.     private String name;
15.
16.     @Transient
17.     @ManyToMany(mappedBy = "roles")
18.     private Set<User> users;
19.
20.     public Role() {
21.     }
22.
23.     public Role(Long id) {
24.         this.id = id;
25.     }
26.
27.     public Role(Long id, String name) {
28.         this.id = id;
29.         this.name = name;

```

```

30.     }
31.
32.     public Long getId() {
33.         return id;
34.     }
35.
36.     public void setId(Long id) {
37.         this.id = id;
38.     }
39.
40.     public String getName() {
41.         return name;
42.     }
43.
44.     public void setName(String name) {
45.         this.name = name;
46.     }
47.
48.     public Set<User> getUsers() {
49.         return users;
50.     }
51.
52.     public void setUsers(Set<User> users) {
53.         this.users = users;
54.     }
55.
56.     @Override
57.     public String getAuthority() {
58.         return getName();
59.     }
60. }
61.

```

### ***UserRepository.java***

```

1. package app.Application.Interfaces;
2.
3. import app.Application.Classes.User;
4. import org.springframework.data.jpa.repository.JpaRepository;
5.
6. public interface UserRepository extends JpaRepository<User, Long> {
7.     User findByUsername(String username);
8. }
9.

```

### ***RoleRepository.java***

```

1. package app.Application.Interfaces;
2.
3. import app.Application.Classes.Role;

```

```

4. import org.springframework.data.jpa.repository.JpaRepository;
5.
6. public interface RoleRepository extends JpaRepository<Role, Long> {
7. }
8.

```

## *UserService.java*

```

1. package app.Application.Services;
2.
3. import app.Application.Classes.Role;
4. import app.Application.Classes.User;
5. import app.Application.Interfaces.RoleRepository;
6. import app.Application.Interfaces.UserRepository;
7. import org.springframework.beans.factory.annotation.Autowired;
8. import org.springframework.security.core.userdetails.UserDetails;
9. import org.springframework.security.core.userdetails.UserDetailsService;
10. import org.springframework.security.core.userdetails.UsernameNotFoundException;
11. import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
12. import org.springframework.stereotype.Service;
13.
14. import javax.persistence.EntityManager;
15. import javax.persistence.PersistenceContext;
16. import java.util.Collections;
17. import java.util.List;
18. import java.util.Optional;
19.
20. @Service
21. public class UserService implements UserDetailsService {
22.     @PersistenceContext
23.     private EntityManager em;
24.
25.     @Autowired
26.     UserRepository userRepository;
27.
28.     @Autowired
29.     RoleRepository roleRepository;
30.
31.     @Autowired
32.     BCryptPasswordEncoder bCryptPasswordEncoder;
33.
34.     public UserService(UserRepository userRepository) {
35.         this.userRepository = userRepository;
36.     }
37.
38.     @Override
39.     public UserDetails loadUserByUsername(String username) throws
        UsernameNotFoundException {
40.         User user = userRepository.findByUsername(username);
41.

```



```

42.         if (user == null) {
43.             throw new UsernameNotFoundException("User not found");
44.         }
45.
46.         return user;
47.     }
48.
49.     public User findById(Long userId) {
50.         Optional<User> userFromDb = userRepository.findById(userId);
51.         return userFromDb.orElse(new User());
52.     }
53.
54.     public List<User> allUsers() {
55.         return userRepository.findAll();
56.     }
57.
58.     public boolean saveUser(User user) {
59.         User userFromDB = userRepository.findByUsername(user.getUsername());
60.
61.         if (userFromDB != null) {
62.             return false;
63.         }
64.
65.         user.setRoles(Collections.singleton(new Role(1L, "ROLE_USER")));
66.         user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
67.         userRepository.save(user);
68.         return true;
69.     }
70.
71.     public boolean deleteUser(Long userId) {
72.         if (userRepository.findById(userId).isPresent()) {
73.             userRepository.deleteById(userId);
74.             return true;
75.         }
76.         return false;
77.     }
78.
79.     public List<User> usergtList(Long idMin) {
80.         return em.createQuery("SELECT u FROM User u WHERE u.id > :paramId",
            User.class)
81.             .setParameter("paramId", idMin).getResultList();
82.     }
83. }
84.

```

## *AdminController.java*

```
1. package app.Application.Controllers;
2.
3. import app.Application.Services.UserService;
4. import org.springframework.beans.factory.annotation.Autowired;
5. import org.springframework.stereotype.Controller;
6. import org.springframework.ui.Model;
7. import org.springframework.web.bind.annotation.GetMapping;
8. import org.springframework.web.bind.annotation.PathVariable;
9. import org.springframework.web.bind.annotation.PostMapping;
10. import org.springframework.web.bind.annotation.RequestParam;
11.
12. @Controller
13. public class AdminController {
14.     @Autowired
15.     private UserService userService;
16.
17.     @GetMapping("/admin")
18.     public String userList(Model model) {
19.         model.addAttribute("allUsers", userService.allUsers());
20.         return "admin";
21.     }
22.
23.     @PostMapping("/admin")
24.     public String deleteUser(@RequestParam(required = true, defaultValue = "" )
        Long userId,
25.                             @RequestParam(required = true, defaultValue = "" )
        String action,
26.                             Model model) {
27.         if (action.equals("delete")){
28.             userService.deleteUser(userId);
29.         }
30.         return "redirect:/admin";
31.     }
32.
33.     @GetMapping("/admin/gt/{userId}")
34.     public String gtUser(@PathVariable("userId") Long userId, Model model) {
35.         model.addAttribute("allUsers", userService.usergtList(userId));
36.         return "admin";
37.     }
38. }
39.
```

## *CookieController.java*

```
1. package app.Application.Controllers;
2.
3. import org.springframework.web.bind.annotation.RestController;
4.
5. @RestController
6. public class CookieController {
7. }
8.
```

## *RegistrationController.java*

```
1. package app.Application.Controllers;
2.
3. import app.Application.Classes.User;
4. import app.Application.Services.UserService;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.stereotype.Controller;
7. import org.springframework.ui.Model;
8. import org.springframework.validation.BindingResult;
9. import org.springframework.web.bind.annotation.GetMapping;
10. import org.springframework.web.bind.annotation.ModelAttribute;
11. import org.springframework.web.bind.annotation.PostMapping;
12.
13. import javax.validation.Valid;
14.
15. @Controller
16. public class RegistrationController {
17.
18.     @Autowired
19.     private UserService userService;
20.
21.     @GetMapping("/registration")
22.     public String registration(Model model) {
23.         model.addAttribute("userForm", new User());
24.
25.         return "registration";
26.     }
27.
28.     @PostMapping("/registration")
29.     public String addUser(@ModelAttribute("userForm") @Valid User userForm,
30. BindingResult bindingResult, Model model) {
31.
32.         if (bindingResult.hasErrors()) {
33.             return "registration";
34.         }
35.         if (!userForm.getPassword().equals(userForm.getPasswordConfirm())){
36.             model.addAttribute("passwordError", "Пароли не совпадают");
37.             return "registration";
38.         }
39.         userService.createUser(userForm);
40.         return "registration";
41.     }
42. }
```

```

37.         }
38.         if (!userService.saveUser(userForm)) {
39.             model.addAttribute("usernameError", "Пользователь с таким именем уже
                существует");
40.             return "registration";
41.         }
42.
43.         return "redirect:/";
44.     }
45. }
46.

```

## *Config.java*

```

1. package app.Application.Configuration;
2.
3. import org.springframework.context.annotation.Configuration;
4. import org.springframework.context.annotation.EnableAspectJAutoProxy;
5. import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
6. import org.springframework.scheduling.annotation.EnableAsync;
7. import
    org.springframework.security.config.annotation.web.configuration.WebSecurityConfigu
        rerAdapter;
8.
9. @Configuration
10. @EnableAspectJAutoProxy
11. @EnableJpaRepositories(basePackages = {"app.Application"})
12. @EnableAsync
13. public class Config extends WebSecurityConfigurerAdapter {
14. }
15.

```

## *MvcConfig.java*

```

1. package app.Application.Configuration;
2.
3. import org.springframework.context.annotation.Configuration;
4. import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
5. import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
6.
7. @Configuration
8. public class MvcConfig implements WebMvcConfigurer {
9.
10.     @Override
11.     public void addViewControllers(ViewControllerRegistry registry) {
12.         registry.addViewController("/login").setViewName("login");
13.         registry.addViewController("/news").setViewName("news");
14.     }
15. }
16.

```

## WebSecurityConfig.java

```
1. package app.Application.Configuration;
2.
3. import app.Application.Services.UserService;
4. import org.springframework.beans.factory.annotation.Autowired;
5. import org.springframework.context.annotation.Bean;
6. import org.springframework.context.annotation.Configuration;
7. import org.springframework.core.annotation.Order;
8. import org.springframework.security.authentication.AuthenticationManager;
9. import
    org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
10. import org.springframework.security.config.annotation.web.builders.HttpSecurity;
11. import
    org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
12. import
    org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
13. import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
14.
15. @Configuration
16. @EnableWebSecurity
17. @Order(1000)
18. public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
19.     @Autowired
20.     UserService userService;
21.     @Bean("authenticationManager")
22.     @Override
23.     public AuthenticationManager authenticationManagerBean() throws Exception {
24.         return super.authenticationManagerBean();
25.     }
26.     @Bean
27.     public BCryptPasswordEncoder bCryptPasswordEncoder() {
28.         return new BCryptPasswordEncoder();
29.     }
30.
31.     @Override
32.     protected void configure(HttpSecurity httpSecurity) throws Exception {
33.         httpSecurity
34.             .csrf().disable().cors().disable()
35.             .authorizeRequests()
36.             .antMatchers("/registration").permitAll()
37.             .anyRequest().authenticated()
38.             .and()
39.             .formLogin()
40.             .loginPage("/login")
41.             .defaultSuccessUrl("/")
42.             .permitAll()
43.             .and()
```

```

44.         .logout().deleteCookies("JSESSIONID")
45.         .permitAll()
46.         .logoutSuccessUrl("/")
47.         .and()
48.         .rememberMe().key("uniqueAndSecret");
49.     }
50.
51.     @Autowired
52.     protected void configureGlobal(AuthenticationManagerBuilder auth) throws
Exception {
53.
54.         auth.userDetailsService(userService).passwordEncoder(bCryptPasswordEncoder());
55.     }
56.

```

## Aspect.java

Код не изменился с предыдущей задачи.

## Результат выполнения программы

```

:: Spring Boot ::
(v2.4.4)

21:15:06.903 [background-prepare] INFO
o.h.validator.internal.util.Version - HV000001: Hibernate Validator 6.1.7.Final
21:15:06.903 [main] INFO
app.Application.Application - Starting Application using Java 11.0.10 on Frischaenn-PC with PID 5916 (C:\Users\frisc\GoogleDrive\Stu
21:15:06.906 [main] INFO
app.Application.Application - No active profile set, falling back to default profiles: default
21:15:07.399 [main] INFO
o.s.d.r.c.RepositoryConfigurationDelegate - Bootstrapping Spring Data JPA repositories in DEFAULT mode.
21:15:07.463 [main] INFO
o.s.d.r.c.RepositoryConfigurationDelegate - Finished Spring Data repository scanning in 55 ms. Found 2 JPA repository interfaces.
21:15:08.174 [main] INFO
o.s.b.w.s.tomcat.TomcatWebServer - Tomcat initialized with port(s): 8081 (http)
21:15:08.182 [main] INFO
o.s.coyote.http11.Http11NioProtocol - Initializing ProtocolHandler ["http-nio-8081"]
21:15:08.182 [main] INFO
o.s.catalina.core.StandardService - Starting service [Tomcat]
21:15:08.182 [main] INFO
o.s.catalina.core.StandardEngine - Starting Servlet engine: [Apache Tomcat/9.0.44]
21:15:08.302 [main] INFO
o.s.a.c.g.C.[Tomcat].[localhost].[/] - Initializing Spring embedded WebApplicationContext
21:15:08.302 [main] INFO
o.s.b.w.s.c.ServletWebServerApplicationContext - Root WebApplicationContext: initialization completed in 1356 ms
21:15:08.445 [main] INFO
o.h.jpa.internal.util.LogHelper - HH000204: Processing PersistenceUnitInfo [name: default]

```

Рисунок 23.1 – Демонстрация работы программы

Please sign in

Username

Password

Sign in

Рисунок 23.2 – Демонстрация работы программы

## Практическая работа №24

### ***Цель работы***

Тема: Тестирование в Spring Framework с использованием Junit.

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

### ***Листинг программы***

#### ***User.java***

Код не изменился с предыдущей задачи.

#### ***Role.java***

Код не изменился с предыдущей задачи.

#### ***UserRepository.java***

Код не изменился с предыдущей задачи.

#### ***RoleRepository.java***

Код не изменился с предыдущей задачи.

#### ***UserService.java***

Код не изменился с предыдущей задачи.

#### ***AdminController.java***

Код не изменился с предыдущей задачи.

#### ***CookieController.java***

Код не изменился с предыдущей задачи.

### *RegistrationController.java*

Код не изменился с предыдущей задачи.

### *Config.java*

Код не изменился с предыдущей задачи.

### *MvcConfig.java*

Код не изменился с предыдущей задачи.

### *WebSecurityConfig.java*

Код не изменился с предыдущей задачи.

### *Aspect.java*

Код не изменился с предыдущей задачи.

### *UserServiceImplTest.java*

```
1. package app.Application.Test;
2.
3. import app.Application.Classes.User;
4. import app.Application.Interfaces.UserRepository;
5. import app.Application.Services.UserService;
6. import org.junit.jupiter.api.Assertions;
7. import org.junit.jupiter.api.Test;
8. import org.junit.jupiter.api.extension.ExtendWith;
9. import org.mockito.Mock;
10. import org.mockito.Mockito;
11. import org.mockito.junit.jupiter.MockitoExtension;
12.
13. import java.util.List;
14.
15. import static org.mockito.Mockito.mock;
16.
17. @ExtendWith(MockitoExtension.class)
18. public class UserServiceImplTest {
19.     @Mock
20.     private UserRepository userRepository;
21.
```



```

22.     @Test
23.     public void getUser() {
24.         userRepository=mock(UserRepository.class);
25.
26.         User user = new User();
27.         user.setUsername("Ivan");
28.
29.         User user1 = new User();
30.         user1.setUsername("Petr");
31.
32.         Mockito.when(userRepository.findAll()).thenReturn(List.of(user,user1));
33.         UserService userService = new UserService(userRepository);
34.         Assertions.assertEquals(2, userService.allUsers().size());
35.         Assertions.assertEquals("Petr",
36.             userService.allUsers().get(0).getUsername());
37.     }
38. }
39.

```

### *Результат выполнения программы*

```

> Task :compileJava UP-TO-DATE
> Task :processResources UP-TO-DATE
> Task :classes UP-TO-DATE
> Task :compileTestJava NO-SOURCE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test NO-SOURCE
BUILD SUCCESSFUL in 1s
2 actionable tasks: 2 up-to-date
21:42:03: Task execution finished 'test'.

```

Рисунок 24.1 – Демонстрация работы программы

## Вывод

В ходе выполнения данных практических работ были получены навыки работы с основными технологиями, необходимыми для создания клиент-серверных приложений. Также были получены навыки работы с фреймворком Spring.

## Список использованных 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: 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/](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/](https://docs.jboss.org/hibernate/stable/search/reference/en-US/html_single/) (дата обращения: 29.01.21). Заголовок с экрана.