Лабораторна робота №16

Модульне тестування

Мета: Розробка модульних тестів з використанням JUnit 5.

1 ВИМОГИ

1. Розробити та додати модульні тести до програм попередніх лабораторних робіт. Забезпечити розділення на рівні початкового коду, тести розташовувати в директоріях з назвою test.

2. Перевірити всі public-методи власного контейнера та його ітератора, які були створені при виконанні завдання лабораторної роботи "9. Параметризація в Java". Забезпечити покриття коду не менше 80%.

3. Перевірити методи, що забезпечують валідацію даних в програмі рішення завдання лабораторної роботи "11. Регулярні вирази. Перевірка даних".

4. Перевірити вирішення прикладної задачі лабораторної роботи "12. Регулярні вирази. Обробка тексту".

5. Перевірити методи обробки контейнера лабораторної роботи "13. Паралельне виконання. Multithreading". Перевіряти тільки обробку даних, виключаючи multithreading (див. п.4).

1.1 Розробник

- П.І.Б: Абдулаєв І. З.
- Группа: КІТ-119в
- Варіант: 1

2 ОПИС ПРОГРАМИ

2.1 Засоби ООП:

Scanner inInt, inStr = new Scanner(System.in) – для введення обраних опцій користувачем з клавіатури;

XMLEncoder encoder = new XMLEncoder(new BufferedOutputStream(new FileOutputStream(filename));

encoder.writeObject(container); – нестандартна серіалізація;

XMLDecoder decoder = new XMLDecoder(new BufferedInputStream(new FileInputStream(filename)));

container = (ArrayList<Challanger>) decoder.readObject(); – нестандартна десеріалізація;

ObjectOutputStream oos = new ObjectOutputStream(new BufferedOutputStream(newFileOutputStream(filename)));

oos.writeObject(container);

oos.flush(); – стандартна серіалізація;

ObjectInputStream ois = new ObjectInputStream(new BufferedOutputStream(new FileInputStream(filename)));

container = (ArrayList<Challanger>) ois.readObject(); – стандартна десеріалізація;

Pattern pattern = Pattern.compile() – компілює регулярний вираз у шаблон;

Matcher matcher = pattern.matcher(data); – створює matcher, який буде відповідати даному вводу для цього шаблону.

assertEquals(expected, actual, "Have to be equals"); - засіб перевірки на еквівалентність тестування.

2.2 Ієрархія та структура класів

Було створено класи 4 Java Unit класів у Source Folder, згідно з звданням, усі методи для тестування починаються з test.

2.3 Важливі фрагменти програми

Клас test09

```
package abdulaev;

import static org.junit.jupiter.api.Assertions.*;

import ua.khpi.oop.abdulaev09.MyContainer;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
```

```
import org.junit.jupiter.params.provider.MethodSource;

class test09 {

    MyContainer<Integer> test_container = new MyContainer<Integer>();

    public static int[] getSizeData() {
        return new int[] { 2, 0, 6};
    }

    @Test
    void testConstructor() {
        MyContainer<Integer> test_container1 = new MyContainer<Integer>();
    }


    @ParameterizedTest
    @MethodSource(value = "getSizeData")
    void testGetSize(int data) {
        int size = data;
        int expected = data;
        test_container.setSize(size);
        int actual = test_container.getSize();
        assertEquals(expected, actual, "Have to be equals");
    }

    @Test
    void testAdd() {
        int expected = 3;
        test_container = add(test_container);
        int actual = test_container.getSize();
        assertEquals(expected, actual, "Have to be equals");
    }

    @Test
    void testDelete() {
        test_container = add(test_container);
        boolean expected1 = true;
        boolean actual1 = test_container.delete(2);
        assertEquals(expected1, actual1, "Have to be the same");
        boolean expected2 = true;
        boolean actual2 = test_container.delete(1);
        assertEquals(expected2, actual2, "Have to be the same");
        boolean expected3 = true;
        boolean actual3 = test_container.delete(0);
        assertEquals(expected3, actual3, "Have to be the same");
        boolean expected4 = false;
        boolean actual4 = test_container.delete(0);
        assertEquals(expected4, actual4, "Have to be the same");
        boolean expected5 = false;
        boolean actual5 = test_container.delete(2);
        assertEquals(expected5, actual5, "Have to be the same");
    }

    @Test
    void testGetElement(){
        test_container = add(test_container);
        Integer expected1 = null;
        Integer actual1 = test_container.getElement(-1);
```

```
        assertEquals(expected1, actual1);
        Integer expected2 = null;
        Integer actual2 = test_container.getElement(60);
        assertEquals(expected2, actual2);
        Integer expected3 = new Integer (-100);
        Integer actual3 = test_container.getElement(2);
        assertEquals(expected3, actual3);
    }

    @Test
    void testIsEmpty() {
        boolean expected_t = true;
        boolean actual_t = test_container.isEmpty();
        assertEquals(expected_t, actual_t, "Have to be equals");
        test_container = add(test_container);
        boolean expected_f = false;
        boolean actual_f = test_container.isEmpty();
        assertEquals(expected_f, actual_f, "Have to be equals");
    }

    @Test
    void testClear() {
        test_container = add(test_container);
        test_container.clear();
        int expected = 0;
        int actual = test_container.getSize();
        assertEquals(expected, actual, "Have to be the same");
    }

    @Test
    void testToString() {
        test_container = add(test_container);
        String expected = "500" + "\n" + "46000" + "\n" + "-100" + "\n";
        String actual = test_container.toString();
        assertEquals(expected, actual, "Have to be the same");
    }

    @Test
    void testToArray(){
        Object[] obj_expected = new Object[3];
        Object[] obj_actual = new Object[3];
        obj_expected[0] = new Integer (500);
        obj_expected[1] = new Integer (46000);
        obj_expected[2] = new Integer (-100);
        test_container = add(test_container);
        obj_actual = test_container.toArray();
        assertEquals(obj_expected[0], obj_actual[0], "Have to be the same");
        assertEquals(obj_expected[1], obj_actual[1], "Have to be the same");
        assertEquals(obj_expected[2], obj_actual[2], "Have to be the same");
    }

    MyContainer<Integer> add(MyContainer<Integer> container){
        Integer a = new Integer (500);
        Integer b = new Integer (46000);
        Integer c = new Integer (-100);
        container.add(a);
        container.add(b);
        container.add(c);
        return container;
```

```
    }
}
```

Класс test11

```
package abdulaev;

import static org.junit.jupiter.api.Assertions.*;

import java.util.regex.Pattern;
import ua.khpi.oop.abdulaev11.RegexCheck;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.MethodSource;

class test11 {

    public static int[][] getData() {
        return new int[][] { {32, 13, 1832, 0}, {0, -10, 2134, 0}, {-4, 11,
2012, 0}, {5, 3, 2018, 1}};
    }

    @ParameterizedTest
    @MethodSource(value = "getData")
    void testIntRegexCheck(int[] data) {
        boolean check = true;
        boolean temp;
        Pattern patternDay = Pattern.compile("([1-9]|[12]\\d|3[01])");
        Pattern patternMonth = Pattern.compile("([1-9]|1[012])");
        Pattern patternYear = Pattern.compile("(19|20)\\d{2}");
        int day = data[0];
        int month = data[1];
        int year = data[2];
        boolean expected = intToBoolean(data[3]);
        temp = RegexCheck.intRegexCheck(day, patternDay);
        check = check & temp;
        temp = RegexCheck.intRegexCheck(month, patternMonth);
        check = check & temp;
        temp = RegexCheck.intRegexCheck(year, patternYear);
        check = check & temp;
        assertEquals(expected, check, "Have to be the same");
    }

    @Test
    void testStringRegexCheck() {
        boolean expected1 = true;
        boolean expected2 = false;
        Pattern patternConditions =
Pattern.compile("(\\w+(\\.|\\s)(\\s|))+");
        String str_true = "Posibility to dose not have buisness trip. Paid
vocations. Free dinner.";
        String str_false = " Paid vocations. Free coffie.  Posibility to have
a nap ";
        boolean actual1 = RegexCheck.stringRegexCheck(str_true,
patternConditions);
        boolean actual2 = RegexCheck.stringRegexCheck(str_false,
patternConditions);
        assertEquals(expected1, actual1, "Have to be the same");
        assertEquals(expected2, actual2, "Havr to be the same");
```

```
    }

    private boolean intToBoolean(int input) {
        if(input == 0) {
            return false;
        }
        else if(input == 1) {
            return true;
        }
        else {
            throw new IllegalArgumentException("Входное значение может быть
равно только 0 или 1 !");
        }
    }
}
```

Class test12

```
package abdulaev;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

import ua.khpi.oop.abdulaev07.Challanger;
import ua.khpi.oop.abdulaev07.DemandsToWork;
import ua.khpi.oop.abdulaev07.WorkExperience;
import ua.khpi.oop.abdulaev10.MyContainer;
import ua.khpi.oop.abdulaev12.Main;

class test12 {

    MyContainer<Challanger> container = new MyContainer<Challanger>();

    @Test
    void testTask() {
        container = generate("actual");
        MyContainer<Challanger> actual = Main.task(container);
        container.delete(4);
        container.delete(3);
        MyContainer<Challanger> expected = container;
        boolean cmp = compare(expected, actual);
        if(cmp) {
            assertEquals(1,1);
        }else {
            assertEquals(0, 1, "Have to be the same.");
        }
    }

    boolean compare(MyContainer<Challanger> expected, MyContainer<Challanger>
actual) {
        String education = null;
        int day = 0;
        int month = 0;
        int year = 0;
        String specializationPrevious = null;
        int experience = 0;
        String specializationNext = null;
        int minSalary = 0;
        String conditions = null;
```

```
        boolean check = true;
        boolean temp = false;
        if(expected.getSize() != actual.getSize()) {
            return !check;
        }
        for(int i = 0; expected.getSize() > i; i++) {
            Challanger element = expected.getElement(i);
            education = element.getEducation();
            day = element.getDismissalDay();
            month = element.getDismissalMonth();
            year = element.getDismissalYear();
            specializationPrevious =
element.getWorkExperience().getSpecialization();
            experience = element.getWorkExperience().getExperience();
            specializationNext =
element.getDemandsToWork().getSpecialization();
            minSalary = element.getDemandsToWork().getMinSalary();
            conditions = element.getDemandsToWork().getConditions();
            Challanger e = actual.getElement(i);
            if(education.equals(e.getEducation())){
                if(day == e.getDismissalDay()) {
                    if(month == e.getDismissalMonth()){
                        if(year == e.getDismissalYear()) {

if(specializationPrevious.equals(e.getWorkExperience().getSpecialization()))
{
                                    if(experience ==
e.getWorkExperience().getExperience()) {

if(specializationNext.equals(e.getDemandsToWork().getSpecialization())) {
                                            if(minSalary ==
e.getDemandsToWork().getMinSalary()) {

if(conditions.equals(e.getDemandsToWork().getConditions())) {
                                                    temp = true;
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
            }
            check = temp & check;
            temp = false;
        }
        return check;
    }

    MyContainer<Challanger> generate(String str){
        WorkExperience workExperienceAdd1 = new WorkExperience("Manager", 5);
        DemandsToWork demandsToWorkAdd1 = new
DemandsToWork("Manager",12600,"Possisbility to dose not have buisness
trip.");
        Challanger challangerAdd1 = new Challanger(container.getSize(),"High
education",31,7,2020,workExperienceAdd1,demandsToWorkAdd1);
        container.add(challangerAdd1);
        WorkExperience workExperienceAdd2 = new WorkExperience("HR-manager",
```

```
15);
        DemandsToWork demandsToWorkAdd2 = new DemandsToWork("HR-
manager",26000,"Free coffie. Possisbility to dose not have buisness trip.");
        Challanger challangerAdd2 = new Challanger(container.getSize(),"High
education",12,12,2012,workExperienceAdd2,demandsToWorkAdd2);
        container.add(challangerAdd2);
        if(str.equals("expected")) {
            return container;
        }
        WorkExperience workExperienceAdd3 = new WorkExperience("Teacher",
47);
        DemandsToWork demandsToWorkAdd3 = new
DemandsToWork("Teacher",26000,"Free coffie. Possisbility to have a nap.");
        Challanger challangerAdd3 = new Challanger(container.getSize(),"High
education",6,4,2021,workExperienceAdd3,demandsToWorkAdd3);
        container.add(challangerAdd3);
        WorkExperience workExperienceAdd4 = new WorkExperience("HR-manager",
2);
        DemandsToWork demandsToWorkAdd4 = new DemandsToWork("HR-
manager",126000,"Free coffie. Possisbility to dose have buisness trip.");
        Challanger challangerAdd4 = new
Challanger(container.getSize(),"School
education",7,8,2018,workExperienceAdd4,demandsToWorkAdd4);
        container.add(challangerAdd4);
        return container;
    }

}
```

Class test13

```
package abdulaev;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

import ua.khpi.oop.abdulaev07.Challanger;
import ua.khpi.oop.abdulaev07.DemandsToWork;
import ua.khpi.oop.abdulaev07.WorkExperience;
import ua.khpi.oop.abdulaev10.MyContainer;
import ua.khpi.oop.abdulaev13.MyThread;

class test13 {

    MyContainer<Challanger> container = new MyContainer<Challanger>();

    @Test
    void testCount() {
        int num = 0;
        addToContainer(num);
        int expected = num;
        MyThread Thread = new MyThread(container, "Thread: 0");
```

```
        int actual = Thread.count();
        Assertions.assertEquals(expected,actual);
    }

    void addToContainer(int num) {
        File file = new File("recruitingAgency11.txt");
        String education1 = null;
        int day1 = 0;
        int month1 = 0;
        int year1 = 0;
        String specializationPrevious1 = null;
        int experience1 = 0;
        String specializationNext1 = null;
        int minSalary1 = 0;
        String conditions1 = null;
        int id1 = 0;
        String education2 = null;
        int day2 = 0;
        int month2 = 0;
        int year2 = 0;
        String specializationPrevious2 = null;
        int experience2 = 0;
        String specializationNext2 = null;
        int minSalary2 = 0;
        String conditions2 = null;
        int id2 =0;
        try {
            Scanner reader = new Scanner(file);
            while(reader.hasNextLine()) {
                String data = reader.nextLine();
                String data1 = reader.nextLine();
                Pattern pattern = Pattern.compile("((\\w+(|\\s))*,\\s([1-
9]|[12]\\d|3[01])\\.([1-9]|1[012])\\.((19|20)\\d{2}),\\s" +
                        "(\\w+.)+,\\s([0-9]|[1-6][0-9]),\\s(\\w+.)+,\\s([1-
9]\\d{3,}),\\s(\\w+(\\.|\\s)(\\s|))+)");
                Matcher matcher = pattern.matcher(data);
                if(matcher.matches()) {
                    String[] information = data.split(",\\s");
                    education1 = information[0];
                    specializationPrevious1 = information[2];
                    experience1 = Integer.parseInt(information[3]);
                    specializationNext1 = information[4];
                    minSalary1 = Integer.parseInt(information[5]);
                    conditions1 = information[6];
                    String[] date1 = information[1].split("\\.");
                    day1 = Integer.parseInt(date1[0]);
                    month1 = Integer.parseInt(date1[1]);
                    year1 = Integer.parseInt(date1[2]);
                }
                Matcher matcher1 = pattern.matcher(data1);
                if(matcher1.matches()) {
                    String[] information1 = data1.split(",\\s");
                    education2 = information1[0];
                    specializationPrevious2 = information1[2];
                    experience2 = Integer.parseInt(information1[3]);
                    specializationNext2 = information1[4];
                    minSalary2 = Integer.parseInt(information1[5]);
                    conditions2 = information1[6];
                    String[] date2 = information1[1].split("\\.");
```

```
                    day2 = Integer.parseInt(date2[0]);
                    month2 = Integer.parseInt(date2[1]);
                    year2 = Integer.parseInt(date2[2]);
                }
            }
            reader.close();
        } catch (FileNotFoundException e){
            e.printStackTrace();
        }
        for(int i = 0; num > i; i++) {
            id1 = container.getSize();
            WorkExperience workExperienceAdd1 = new
WorkExperience(specializationPrevious1, experience1);
            DemandsToWork demandsToWorkAdd1 = new
DemandsToWork(specializationNext1,minSalary1,conditions1);
            Challanger challangerAdd1 = new
Challanger(id1++,education1,day1,month1,year1,workExperienceAdd1,demandsToWor
kAdd1);
            container.add(challangerAdd1);
            id2 = container.getSize();
            WorkExperience workExperienceAdd2 = new
WorkExperience(specializationPrevious2, experience2);
            DemandsToWork demandsToWorkAdd2 = new
DemandsToWork(specializationNext2,minSalary2,conditions2);
            Challanger challangerAdd2 = new
Challanger(id2++,education2,day2,month2,year2,workExperienceAdd2,demandsToWor
kAdd2);
            container.add(challangerAdd2);
        }
    }
}
```

3 Результати роботи програми

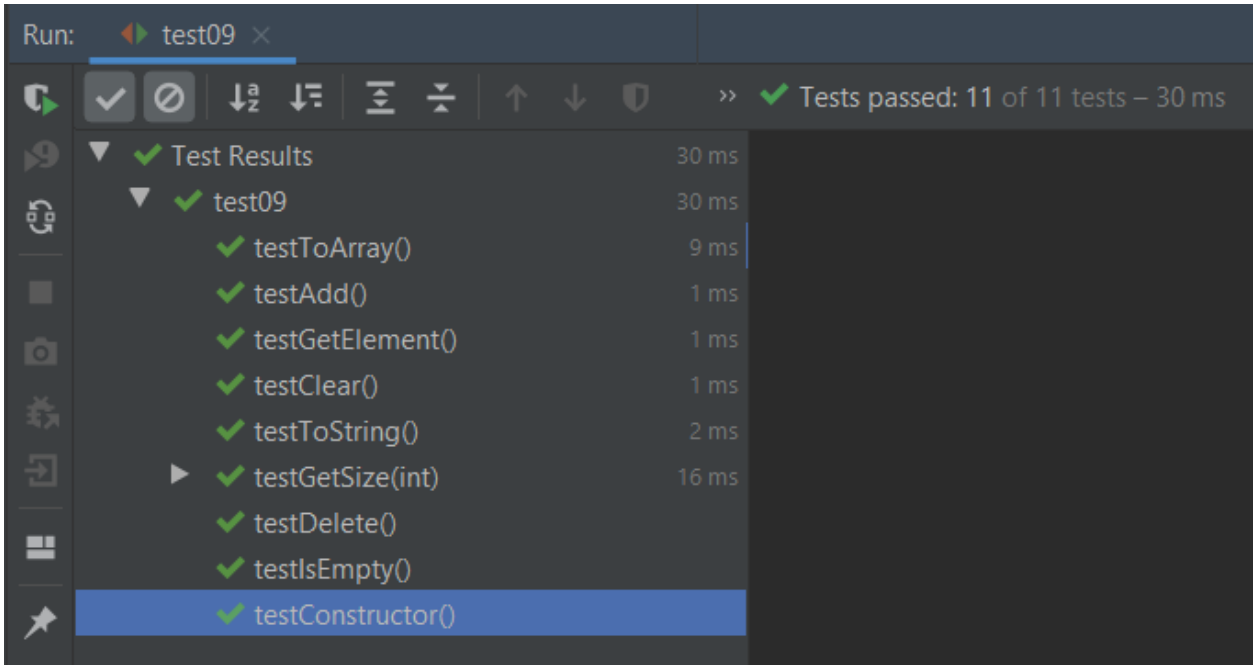

Рисунок 16.1 – Результат тестування 9 ЛР

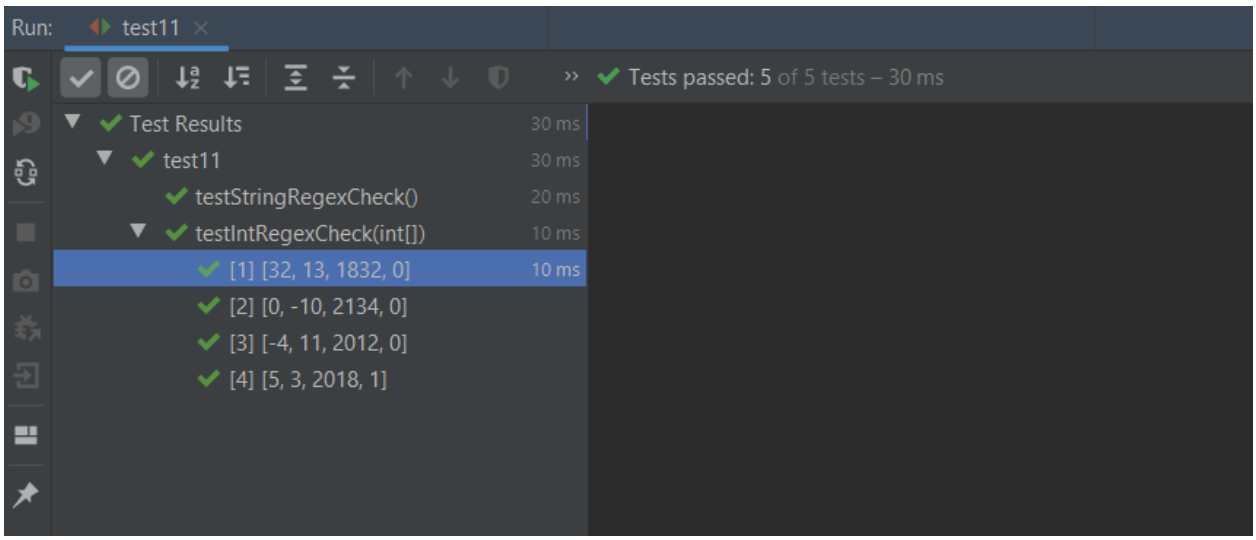

Рисунок 16.2 – Результат тестування 11 ЛР

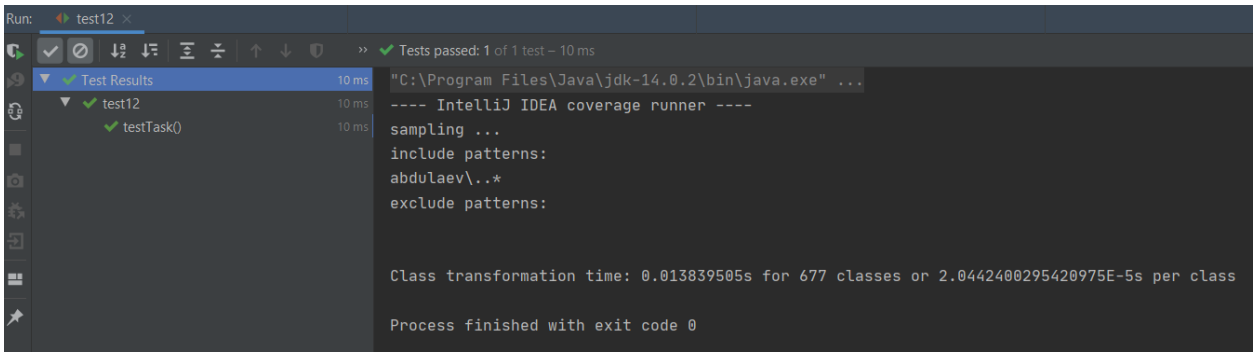

Рисунок 16.3 – Результат тестування 12 ЛР

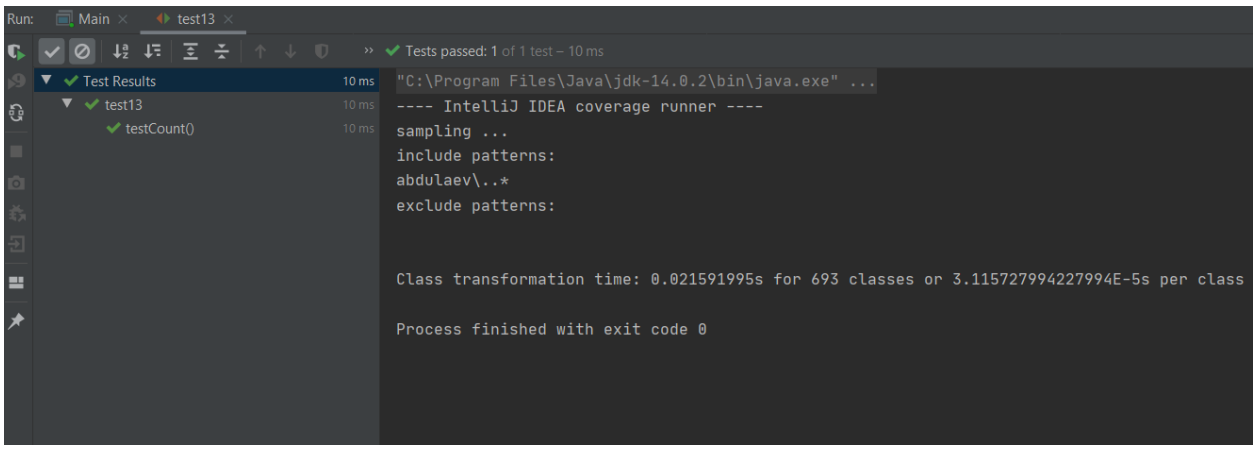

Рисунок 16.4 – Результат тестування 13 ЛР

Висновок

Під час виконання лабораторної роботи було набуто навички роботи з модульними тестами з використанням JUnit 5 в середовищі IntelliJ IDEA.