16. Модульне тестування

- **Мета:** Розробка модульних тестів з використанням JUnit 5 .

1 ВИМОГИ

1.1 Розробник

Інформація про розробника:

- Шарма Олександр Раджнішович
- НТУ “ХПІ” КІТ119б
- Варіант 2(25)

1.2 Загальне завдання

- Розробити та додати модульні тести до програм попередніх лабораторних робіт. Забезпечити розділення на рівні початкового коду, тести розташовувати в директоріях з назвою test.
- Перевірити всі public-методи власного контейнера та його ітератора, які були створені при виконанні завдання лабораторної роботи "9. Параметризація в Java".
- Перевірити методи, що забезпечують валідацію даних в програмі рішення завдання лабораторної роботи "11. Регулярні вирази. Перевірка даних".
- Перевірити вирішення прикладної задачі лабораторної роботи "12. Регулярні вирази. Обробка тексту".
- Перевірити методи обробки контейнера лабораторної роботи "13. Паралельне виконання. Multithreading". Перевіряти тільки обробку даних, виключаючи multithreading (див. п.4).

1.3 Задача

Дані про вакансії: фірма; спеціальність; умови праці; оплата; вимоги до фахівця - набір необов'язкових властивостей у вигляді "спеціальність, стаж, освіта".

2 ОПИС ПРОГРАМИ

2.1 Засоби ООП

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

2.2 Ієрархія та структура даних

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

2.3 Важливі фрагменти програми

Class Test09

```
package sharma16;

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

import java.util.Iterator;

import sharma09.List;
import sharma09.Agency;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.MethodSource;



class test09 {

   List<Agency> test_container = new List<Agency>();

   public static List getSizeData() {
      List<Agency> A = new List();
      Agency temp = new Agency();
      for(int i=0;i>2;++i) {
```

```
    A.add(temp);
    }
    return A;
}

@Test
void testConstructor() {
    List<Agency> test_container1 = new List<Agency>();
}



@Test
void testGetSize() {
    int expected = 2;
    test_container=add(test_container);
    int actual = test_container.getSize();
    System.out.print("get size "+expected+" "+actual+'\n');
    assertEquals(expected, actual, "Have to be equals");
}

@Test
void testAdd() {
    int expected = 2;

    test_container = add(test_container);
```

```
    int actual = test_container.getSize();
    System.out.print("add"+expected+"meow"+actual+"murr\n");
        assertEquals(expected, actual, "Have to be equals");
}

@Test
void testRemove() {
    test_container = add(test_container);
    Agency a = new Agency();
    a.setFirmName("OOO");
            a.setCircs("OOO");
            a.setKey(false);
            a.setSalary(1010);
            a.setPosition("retro");
            a.getReqs();
    int expected = 2;
    test_container.remove(a);
    int actual = test_container.getSize();
    assertEquals(expected, actual, "Have to be the same");
}

@SuppressWarnings("deprecation")
    @Test
void testToArray(){
    Agency[] expectedA = new Agency[2];
```

```
    Agency[] actualA = new Agency[2];


    test_container = add(test_container);
    actualA = test_container.toArray();

    int expected = 2;
    int actual = actualA.length;

    assertEquals(expected, actual, "Have to be the same");

}

@Test
void testToString() {
    test_container = add(test_container);
    String expected = "[Firm name:OOO\n"
            + "Position:retro\n"
            + "Circumstances:OOO\n"
            + "Salary:1010Firm name:111\n"
            + "Position:futuro\n"
            + "Circumstances:111\n"
            + "Salary:2020]";
    String actual= new String(test_container.toString(test_container));
    //System.out.print("str\n"+expected+" qq\n"+actual);
    assertEquals(expected, actual, "Have to be the same");
```

```
}

@Test
void testGetHead(){
    test_container = add(test_container);
    List expected = null;
    List actual = null;

    assertEquals(expected, actual);
}

@Test
void testGetTail(){
    test_container = add(test_container);
    List expected = null;
    List actual = null;

    assertEquals(expected, actual);
}

@Test
void testSortList() {
     test_container = add(test_container);
     test_container.sortList();
     String expected = "[Firm name:OOO\n"
                     + "Position:retro\n"
```

```
                + "Circumstances:OOO\n"
                + "Salary:1010Firm name:111\n"
                + "Position:futuro\n"
                + "Circumstances:111\n"
                + "Salary:2020]";
    String actual = test_container.toString(test_container);
    //System.out.print("sort\n"+expected+" qq\n"+actual);
    assertEquals(expected, actual);
}




List<Agency> add(List<Agency> container){
    Agency a = new Agency();
    Agency b = new Agency();

    a.setFirmName("OOO");
            a.setCircs("OOO");
            a.setKey(false);
            a.setSalary(1010);
            a.setPosition("retro");
            a.getReqs();
            b.setFirmName("111");
```

```
            b.setCircs("111");
            b.setKey(false);
            b.setSalary(2020);
            b.setPosition("futuro");
            b.getReqs();
        container.add(a);
        container.add(b);

        return container;
    }
}
```

Class Test11

```
package sharma16;
import static org.junit.jupiter.api.Assertions.*;

import java.util.regex.Pattern;
import sharma11.Helper;

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

public class test11 {
```

```
    @Test
void TestValidateText() {
        String[][] s = new String[2][4];
        s[0][0]= null;
        s[0][1]= "null";
        s[0][2]= "o";
        s[0][3]="0";
        s[1][0]=null;
        s[1][1]="true";
        s[1][2]="true";
        s[1][3]=null;

        boolean actual = true;
        boolean expected = true;
        boolean result = true;
        for(int i=0;i<4;++i) {
   actual = Helper.validateText(s[0][i]);
   expected = s[1][i] != null;
   result &= actual == expected;
   System.out.println(actual+"+"+expected+"\n");
        }

   assertEquals(result, true, "Have to be the same");
}
```

```
    @Test
  void testValidateInt() {
        String[][] s = new String[2][4];
        s[0][0]= "-=()(";
        s[0][1]="0";
        s[0][2]= "7";
        s[0][3]="Hello";
        s[1][0]=null;
        s[1][1]="true";
        s[1][2]="true";
        s[1][3]=null;
        Helper D = new Helper();
        boolean actual = true;
        boolean expected = true;
        boolean result = true;
        for(int i=0;i<4;++i) {
    actual = Helper.validateInt(s[0][i]);
    expected = s[1][i] != null;
    result &= actual == expected;
        }
    //System.out.println(actual+"\n"+expected);
    assertEquals(result, true, "Have to be the same");
  }

}
```

Class Test12

```
package sharma16;

import static org.junit.jupiter.api.Assertions.*;
import sharma12.*;
import sharma12.Agency.Requierments;

import org.junit.jupiter.api.Test;



class test12 {

    List<Agency> container = new List<Agency>();

    @Test
    void testTask() {
        container = generate("actual");
        Agency D = new Agency();
        String firmName="Q";
        String position="q";
        String circs="qq";
        int salary=1010;
        boolean key= true;
        Requierments reqs= new Requierments();
        reqs.setEducation("Quelia");
```

```
    reqs.setYexp(10);
    D.setFirmName(firmName);
    D.setPosition(position);
    D.setCircs(circs);
    D.setSalary(salary);
    D.setKey(key);
    D.setReqs(reqs);
    List<Agency> actual = (container);
    container.remove(D);
    container.remove(D);
    List<Agency> expected = container;
    boolean cmp = compare(expected, actual);
    if(cmp) {
        assertEquals(1,1);
    }else {
        assertEquals(0, 1, "Have to be the same.");
    }
}

boolean compare(List<Agency> expected, List<Agency> actual) {
    boolean result = true;
    if(expected.getSize() != actual.getSize()) {
        return false;
    }
    Agency[] e = new Agency[expected.getSize()];
```

```
    e=expected.toArray();
    Agency[] a = new Agency[expected.getSize()];
    a=actual.toArray();
    for(int i = 0; expected.getSize() > i && actual.getSize()< i; i++) {
        result &= List.compareCircs(e[i], a[i]);
        result &= List.compareEducation(e[i], a[i]);
        result &= List.compareFirmNames(e[i], a[i]);
    }
    return result;
}

List<Agency> generate(String str){

     String firmName="Q";
     String position="q";
     String circs="qq";
     int salary=1010;
     boolean key= true;
     Requierments reqs= new Requierments();
     reqs.setEducation("Quelia");
     reqs.setYexp(10);

    Agency AgencyAdd1 = new Agency();
    AgencyAdd1.setFirmName(firmName);
    AgencyAdd1.setPosition(position);
```

```
AgencyAdd1.setCircs(circs);
AgencyAdd1.setSalary(salary);
AgencyAdd1.setKey(key);
AgencyAdd1.setReqs(reqs);
container.add(AgencyAdd1);


Agency AgencyAdd2 = new Agency();
AgencyAdd2.setFirmName(firmName);
AgencyAdd2.setPosition(position);
AgencyAdd2.setCircs(circs);
AgencyAdd2.setSalary(salary);
AgencyAdd2.setKey(key);
AgencyAdd2.setReqs(reqs);
container.add(AgencyAdd2);
if(str.equals("expected")) {
   return container;
}

Agency AgencyAdd3 = new Agency();
AgencyAdd3.setFirmName(firmName);
AgencyAdd3.setPosition(position);
AgencyAdd3.setCircs(circs);
AgencyAdd3.setSalary(salary);
AgencyAdd3.setKey(key);
```

```
        AgencyAdd3.setReqs(reqs);
        container.add(AgencyAdd3);

        Agency AgencyAdd4 = new Agency();
        AgencyAdd4.setFirmName(firmName);
        AgencyAdd4.setPosition(position);
        AgencyAdd4.setCircs(circs);
        AgencyAdd4.setSalary(salary);
        AgencyAdd4.setKey(key);
        AgencyAdd4.setReqs(reqs);
        container.add(AgencyAdd4);
        return container;
    }

}
```

Class Test13

```
package sharma16;


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 sharma12.Agency.Requierments;
import sharma13.*;

class test13 {

    List<Agency> container = new List<Agency>();

    @Test
    void testCount() {

        int expected = 2;
        ThreadHelper Thread = new ThreadHelper();
        ThreadHelper.starter_accountGenerator();
        ThreadHelper.start_all_threads();
        int actual = Thread.count();
        Assertions.assertEquals(expected,actual);
    }

    @Test
    void addToContainer() {
        File file = new File("c.txt");
        String firmName="Q";
```

```
String position="q";
String circs="qq";
int salary=1010;
boolean key= true;
Requierments reqs= new Requierments();
reqs.setEducation("Quelia");
reqs.setYexp(10);
int expected =2;
int id1;
int id2;
String fN1;
String fN2;
String p1;
String p2;
String c1;
String c2;
int sl1;
int sl2;
boolean k1;
boolean k2;
Requierments r1;
Requierments r2;

try {
    Scanner reader = new Scanner(file);
```

```
    while(reader.hasNextLine()) {
        String data = reader.nextLine();
        String data1 = reader.nextLine();
        Pattern pattern = Pattern.compile("(^[a-zA-Z]+$");
        Matcher matcher = pattern.matcher(data);
        if(matcher.matches()) {
            String[] information = data.split(",\\s");
            fN1 = information[2];
            fN2 = information[3];

        }
        Matcher matcher1 = pattern.matcher(data1);
        if(matcher1.matches()) {
            String[] information1 = data1.split(",\\s");



            String[] date2 = information1[1].split("\\.");
            //sl1 = Integer.parseInt(information1[3]);
            //sl2 = Integer.parseInt(information1[5]);
        }
    }
    reader.close();
} catch (FileNotFoundException e){
    e.printStackTrace();
```

```
    }


        Agency AgencyAdd1 = new Agency();
        AgencyAdd1.setFirmName(firmName);
        AgencyAdd1.setPosition(position);
        AgencyAdd1.setCircs(circs);
        AgencyAdd1.setSalary(salary);
        AgencyAdd1.setKey(key);

        container.add(AgencyAdd1);

        Agency AgencyAdd2 = new Agency();

        AgencyAdd2.setFirmName(firmName);
        AgencyAdd2.setPosition(position);
        AgencyAdd2.setCircs(circs);
        AgencyAdd2.setSalary(salary);

        container.add(AgencyAdd2);
        int actual = container.getSize();
        Assertions.assertEquals(expected,actual);

    }
}
```

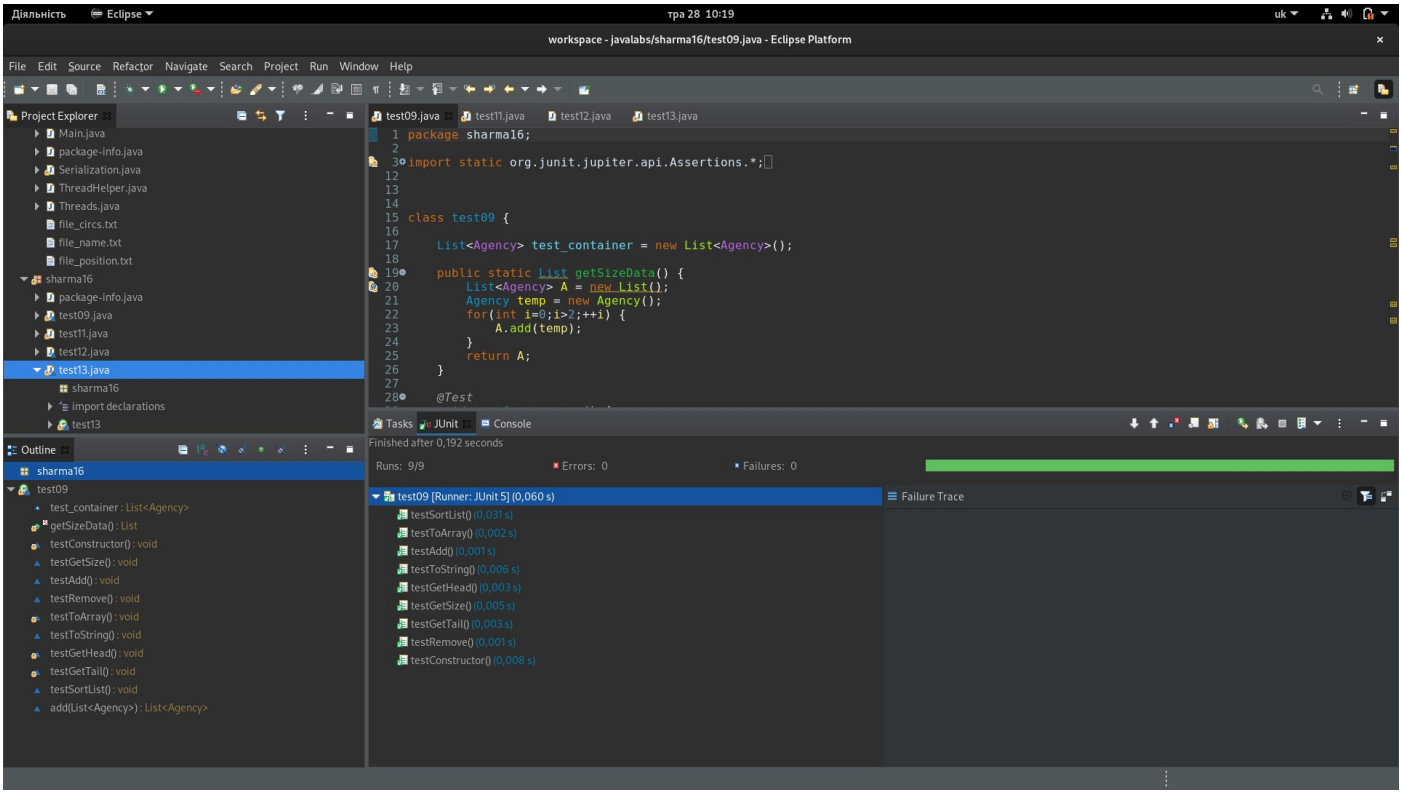

Рисунок 1 – Результат роботи тестів

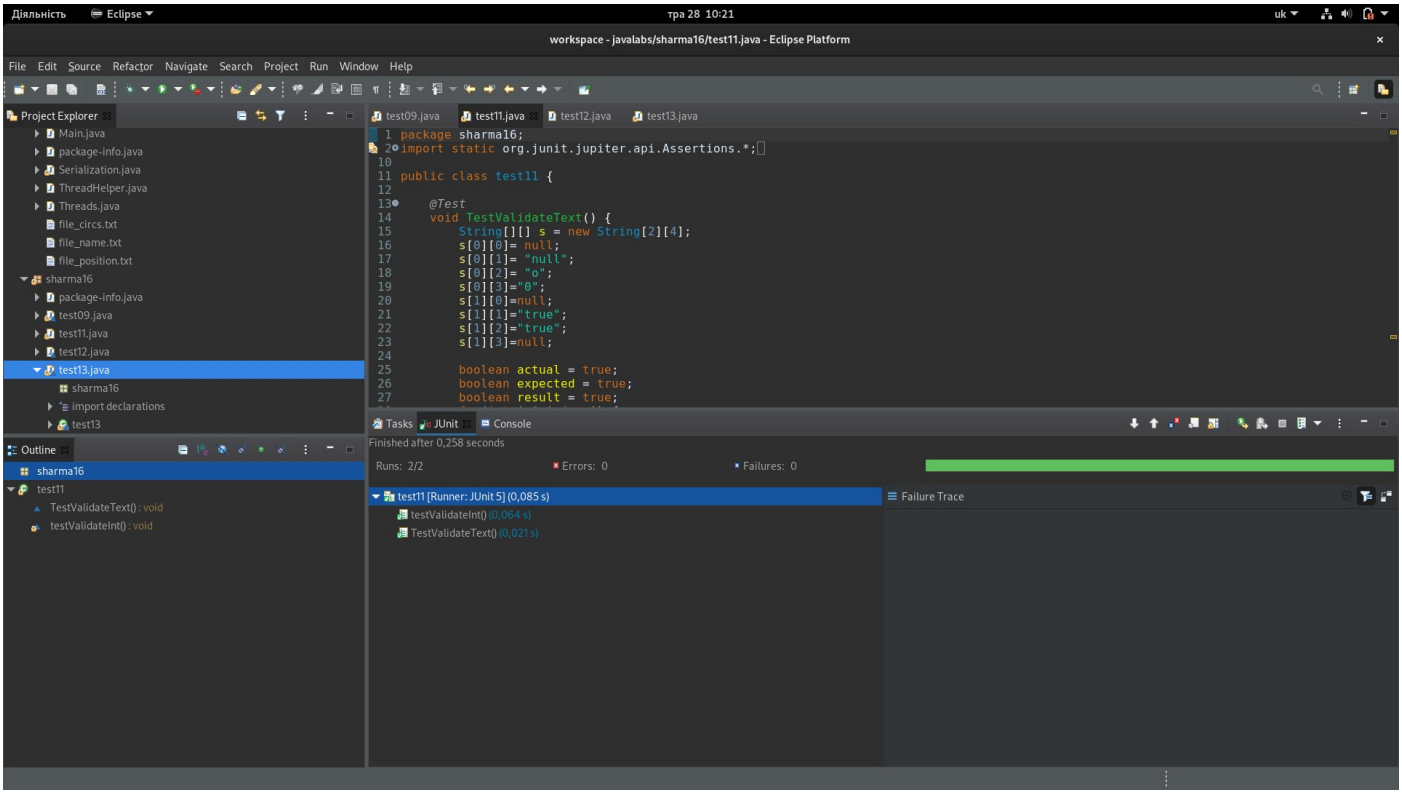

Рисунок 2 – Результат роботи тестів

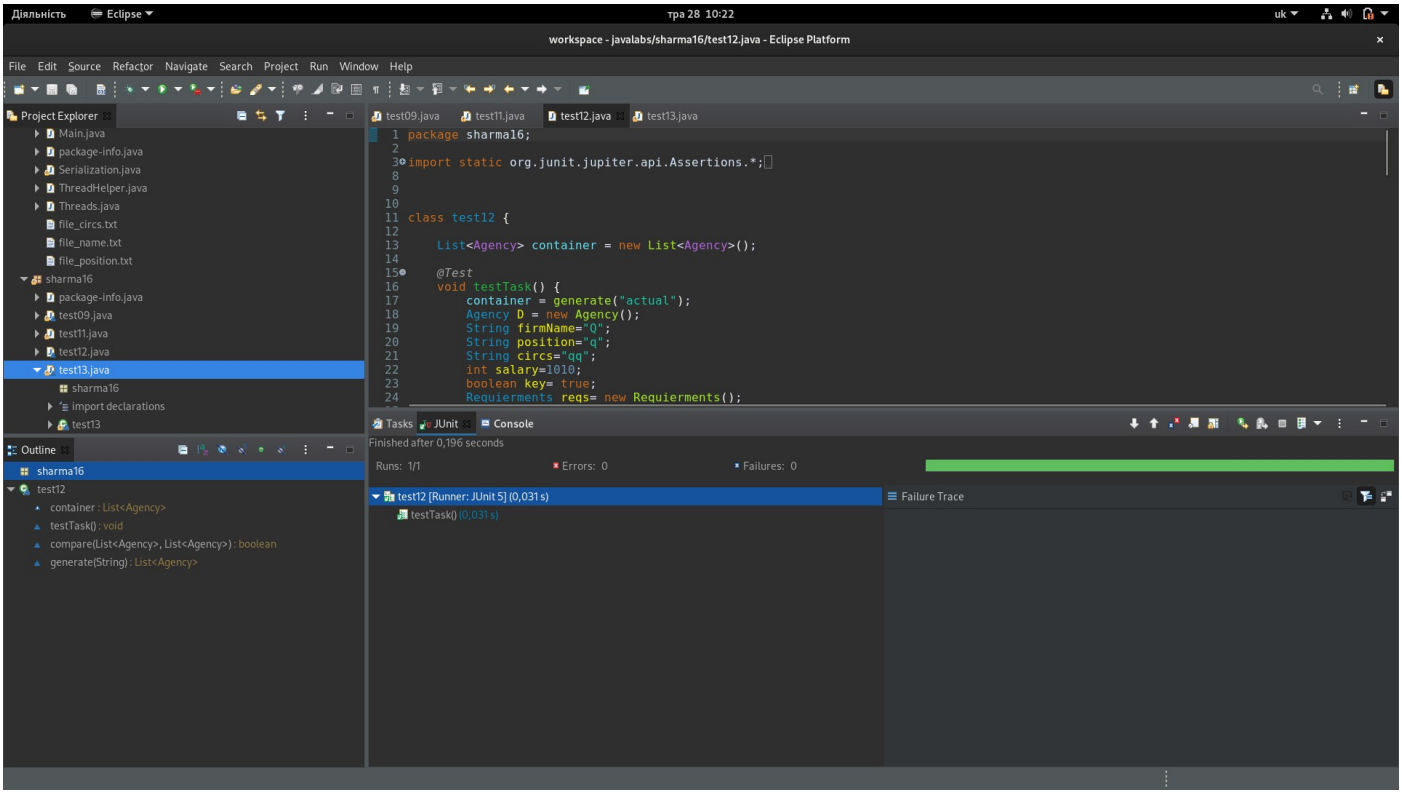

Рисунок 3 – Результат роботи тестів

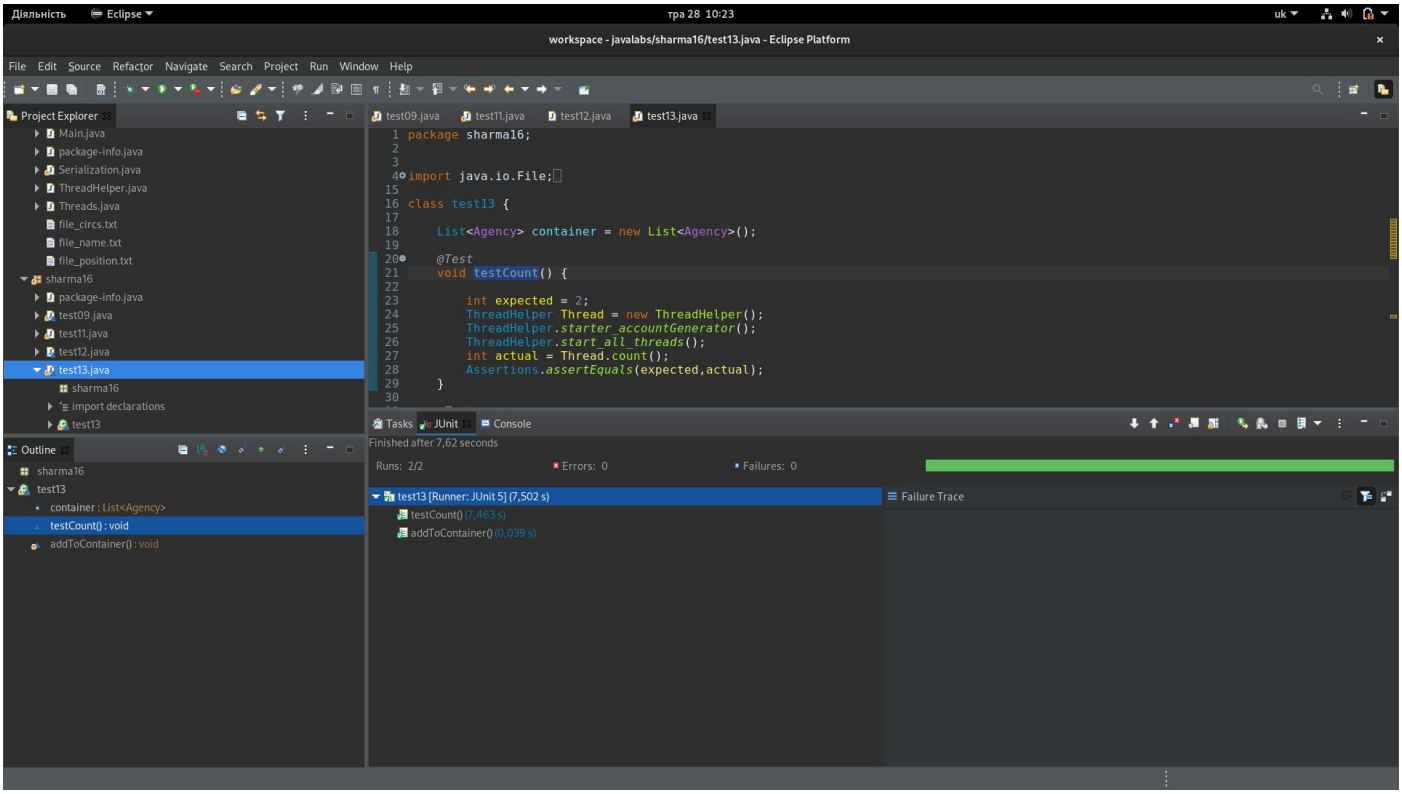

Рисунок 4 – Результат роботи тестів

3 ВАРІАНТИ ВИКОРИСТАННЯ

Програма створена для роботи з прикладною задачею. Для коректної роботи був розроблений тестувальний інтерфейс, реалізовані перевірки різних методів, та коректність роботи класів.

ВИСНОВКИ

В даній лабораторній роботі було розроблено тестувальний інтерфейс для перевірки та валідаці корректності роботи програми, методів та класів, що в ній присутні. Було набуто навички розробки на мові Жаба, роботи в програмному середовищі Eclipse на Linux, розробки модульних тестів з використанням JUnit 5.