Министерство науки и высшего образования Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

**(ВлГУ)**

**Кафедра информационных систем и программной инженерии**

Лабораторная работа №7

по дисциплине «Платформонезависимое

программирование»

Тема: «Автоматизированное тестирование программных модулей»

Выполнил:

студент гр. ПРИ-123

А.Ц. Нямаа

Приняла:

Ст. пр. кафедры ИСПИ

О.Н. Шамышева

Владимир, 2025 г.

ЦЕЛЬ РАБОТЫ:

Получить навыки работы с модулем JUnit.

ЗАДАНИЕ 1 (вариант 12).

Пример: необходимо реализовать сортировку массива чисел с помощью сортировки подсчетом. Гарантируется, что все числа в массиве будут в диапазоне от 0 до 1000. Гарантируется, что длина массива не превысит 106. Необходимо произвести тестирование на корректность, а также нагрузочное тестирование данного метода.

ВЫПОЛНЕНИЕ РАБОТЫ:

В начале выполнения данного задания я определил физическую структуру приложения.

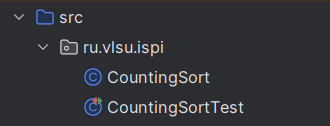


Рисунок 1. Физическая структура программы.

Далее написал следующие классы: CountingSort, представляющий реализацию сортировки массива чисел с помощью сортировки подсчетом, и CountingSortTest, который включает набор тестов согласно требованиям из методического указания.

Листинг CountingSort.java:

package ru.vlsu.ispi;  
  
public class CountingSort {  
 private final int[] aux = new int[1001];  
  
 public int[] sort(int[] arr){  
 int[] result = new int[arr.length];  
 int pos = 0;  
  
 for (int i : arr){  
 aux[i]++;  
 }  
  
 for (int i = 0; i <= 1000; i++){  
 while (aux[i] > 0){  
 result[pos] = i;  
 pos++;  
 aux[i]--;  
 }  
 }  
  
 return result;  
 }  
}

Листинг CountingSortTest.java:

package ru.vlsu.ispi;  
  
import org.junit.Test;  
import java.util.Arrays;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class CountingSortTest {  
 CountingSort countingSort = new CountingSort();  
  
 @Test  
 public void SimpleTest(){  
 int[] arrToTest = {1, 5, 2};  
 int[] expectingArr = {1, 2, 5};  
  
 assertArrayEquals(expectingArr, countingSort.sort(arrToTest));  
 }  
  
 @Test  
 public void ReverseTest(){  
 int arrLength = 100;  
  
 int[] arrToTest = new int[arrLength];  
 int[] expectingArr = new int[arrLength];  
 for (int i = 0; i < arrLength; i++){  
 arrToTest[i] = 99 - i;  
 expectingArr[i] = i;  
 }  
 assertArrayEquals(expectingArr, countingSort.sort(arrToTest));  
 }  
  
 @Test  
 public void RandomTest(){  
 int arrLength = 100;  
  
 int[] arrToTest = new int[arrLength];  
 int[] expectingArr = new int [arrLength];  
  
 for (int i = 0; i < arrLength; i++){  
 arrToTest[i] = (int) (Math.random() \* 100);  
 expectingArr[i] = arrToTest[i];  
 }  
  
 Arrays.sort(expectingArr);  
 assertArrayEquals(expectingArr, countingSort.sort(arrToTest));  
 }  
  
 private int[] generate(int length){  
 int[] arrToGenerate = new int[length];  
 for (int i = 0; i < length; i++){  
 arrToGenerate[i] = (int) (Math.random() \* 100);  
 }  
 return arrToGenerate;  
 }  
  
 private void check(int[] arrToTest){  
 for (int i = 0; i < arrToTest.length - 1; i++){  
 assertTrue(arrToTest[i] <= arrToTest[i + 1]);  
 }  
 }  
  
 @Test  
 public void StressTestOne(){  
 check(countingSort.sort(generate(1000)));  
 }  
  
 @Test  
 public void StressTestSecond(){  
 check(countingSort.sort(generate(10000)));  
 }  
  
 @Test  
 public void StressTestThird(){  
 check(countingSort.sort(generate(1000000)));  
 }  
}

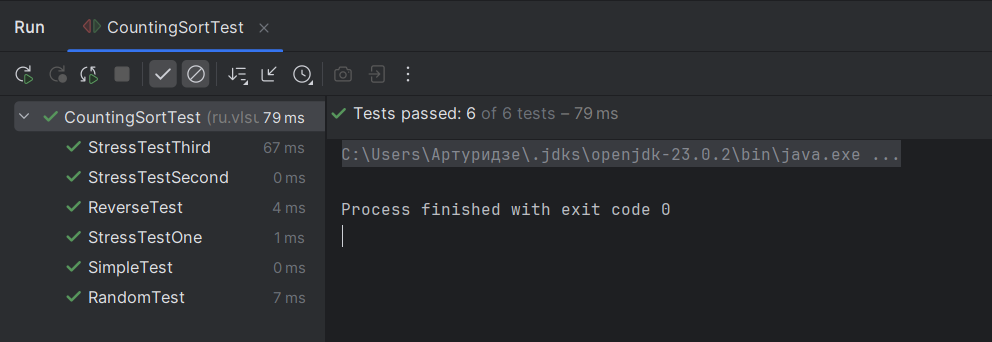


Рисунок 2. Выполнение набора тестов для проверки созданного приложения.

ЗАДАНИЕ 2 (вариант 12).

Необходимо разработать класс, который производит сортировку массива целых чисел с помощью быстрой сортировки. Также необходимо разработать следующие наборы тестов: небольшие тесты, которые проверяют функцию на корректность, тесты для нагрузочного тестирования (сгенерировать массив очень большого размера) и тесты, выполняющие функцию последовательно много раз для различных значений аргументов, чтобы получить среднее время выполнение такой функции.

ВЫПОЛНЕНИЕ РАБОТЫ:

В начале выполнения данного задания я определил физическую структуру приложения.

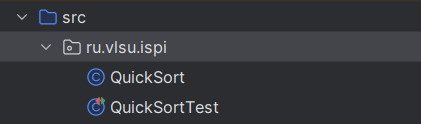


Рисунок 3. Физическая структура программы.

Далее написал некоторые классы, необходимые для выполнения данной работы. Класс QuickSort выполняет сортировку при помощи метода, названного быстрой сортировкой, и класс QuickSortTest содержит набор тестов, необходимых по условию задачи.

Листинг QuickSort.java:

package ru.vlsu.ispi;  
  
public class QuickSort {  
 public int[] sort(int[] arrToSort, int low, int high){  
 if (arrToSort.length == 0 || low >= high){  
 return arrToSort;  
 }  
  
 int middle = low + (high - low) / 2;  
 int border = arrToSort[middle];  
  
 int i = low, j = high;  
 while (i <= j){  
 while (arrToSort[i] < border){  
 i++;  
 }  
 while (arrToSort[j] > border){  
 j--;  
 }  
 if (i <= j){  
 int swap = arrToSort[i];  
 arrToSort[i] = arrToSort[j];  
 arrToSort[j] = swap;  
 i++;  
 j--;  
 }  
 }  
  
 if (low < j){  
 sort(arrToSort, low, j);  
 }  
 if (high > i){  
 sort(arrToSort, i, high);  
 }  
  
 return arrToSort;  
 }  
}

Листинг QuickSortTest.java:

package ru.vlsu.ispi;  
  
import org.junit.Test;  
import java.util.Arrays;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class QuickSortTest {  
 QuickSort quickSort = new QuickSort();  
  
 @Test  
 public void SimpleTest(){  
 int[] arrToTest = {1, 5, 2};  
 int[] expectingArr = {1, 2, 5};  
  
 assertArrayEquals(expectingArr, quickSort.sort(arrToTest, 0, arrToTest.length - 1));  
 }  
  
 @Test  
 public void ReverseTest(){  
 int arrLength = 100;  
  
 int[] arrToTest = new int[arrLength];  
 int[] expectingArr = new int[arrLength];  
 for (int i = 0; i < arrLength; i++){  
 arrToTest[i] = 99 - i;  
 expectingArr[i] = i;  
 }  
 assertArrayEquals(expectingArr, quickSort.sort(arrToTest, 0, arrToTest.length - 1));  
 }  
  
 @Test  
 public void RandomTest(){  
 int arrLength = 100;  
  
 int[] arrToTest = new int[arrLength];  
 int[] expectingArr = new int [arrLength];  
  
 for (int i = 0; i < arrLength; i++){  
 arrToTest[i] = (int) (Math.random() \* 100);  
 expectingArr[i] = arrToTest[i];  
 }  
  
 Arrays.sort(expectingArr);  
 assertArrayEquals(expectingArr, quickSort.sort(arrToTest, 0, arrToTest.length - 1));  
 }  
  
 private int[] generate(int length){  
 int[] arrToGenerate = new int[length];  
 for (int i = 0; i < length; i++){  
 arrToGenerate[i] = (int) (Math.random() \* 100);  
 }  
 return arrToGenerate;  
 }  
  
 private void check(int[] arrToTest){  
 for (int i = 0; i < arrToTest.length - 1; i++){  
 assertTrue(arrToTest[i] <= arrToTest[i + 1]);  
 }  
 }  
  
 @Test  
 public void StressTestOne(){  
 int[] arrToTest = generate(10000);  
 check(quickSort.sort(arrToTest, 0, arrToTest.length - 1));  
 }  
  
 @Test  
 public void StressTestSecond(){  
 int[] arrToTest = generate(100000);  
 check(quickSort.sort(arrToTest, 0, arrToTest.length - 1));  
 }  
  
 @Test  
 public void StressTestThird(){  
 int[] arrToTest = generate(1000000);  
 check(quickSort.sort(arrToTest, 0, arrToTest.length - 1));  
 }  
  
 @Test  
 public void AverageTimeTest(){  
 int testsCount = 10000;  
 long[] timeList = new long[testsCount];  
  
 for (int i = 0; i < testsCount; i++){  
 long startTime = System.currentTimeMillis();  
  
 int[] arrToTest = generate(100000);  
 quickSort.sort(arrToTest, 0, arrToTest.length - 1);  
  
 long endTime = System.currentTimeMillis();  
  
 timeList[i] = endTime - startTime;  
 }  
  
 long totalTime = 0;  
 for (long time : timeList){  
 totalTime += time;  
 }  
  
 double averageTime = (double) totalTime / testsCount;  
 System.out.println("Среднее время выполнения: " + averageTime + " миллисекунд");  
 }  
}

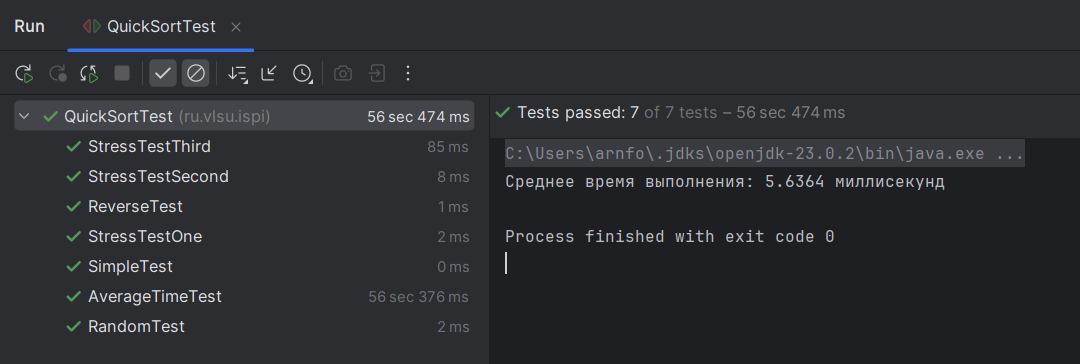


Рисунок 4. Выполнение набора тестов для проверки созданного приложения.

ВЫВОД К РАБОТЕ:

Я получил навыки работы с модулем JUnit.