

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет
по лабораторной работе № 1 по дисциплине «Тестирование программного
обеспечения»

Автор: Иванов Андрей Вячеславович

Факультет: ПИиКТ

Группа: Р33101

Преподаватель: Машина Екатерина Алексеевна



Санкт-Петербург, 2024

Задание

Лабораторная работа #1

1. Для указанной функции провести модульное тестирование разложения функции в степенной ряд. Выбрать достаточное тестовое покрытие.
2. Провести модульное тестирование указанного алгоритма. Для этого выбрать характерные точки внутри алгоритма, и для предложенных самостоятельно наборов исходных данных записать последовательность попадания в характерные точки. Сравнить последовательность попадания с эталонной.
3. Сформировать доменную модель для заданного текста. Разработать тестовое покрытие для данной доменной модели

Введите вариант:

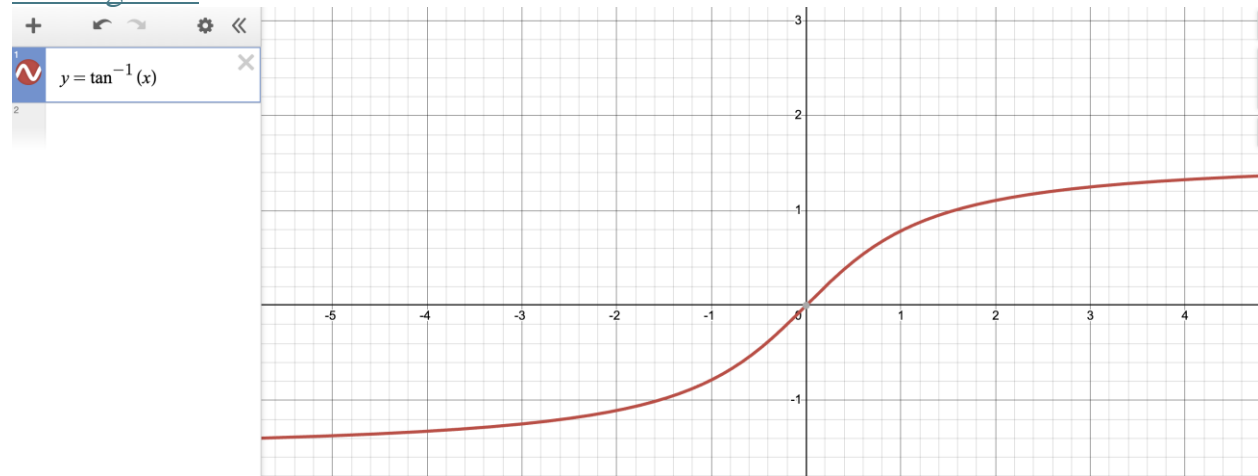
1. Функция $\arctg(x)$
2. Программный модуль для сортировки массива по алгоритму быстрой сортировки
(<http://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html>)
3. Описание предметной области:

Через несколько тысяч лет, когда их галактика уже лежала в руинах, они поняли, в конце концов, что все это было ужасной ошибкой, и тогда оба враждующих боевых флота объединили свои остатки с тем, чтобы совершить совместное нападение на нашу Галактику, положительно определенную как источник обидной фразы.

Выполнение

Исходники:

<https://github.com/ANVISERO/ITMO/tree/main/3course/2semester/Software%20Testing/lab1>



Реализация функции $\arctg(x)$:

```
package com.anvisero.task1;

public class ArcTg {
    public static double calculate(double x) {
        if (x < -1 || x > 1) {
            return Double.NaN;
        }
        double result = 0;
        double numerator = x;
        int substitute = 1;
        int n = 0;
        double accuracy = 1;
        while (Math.abs(accuracy) >= 1.0E-5) {
            result += Math.pow(-1, n) * numerator / substitute;
            n++;
            numerator = numerator * x * x;
            substitute += 2;
        }
    }
}
```

```

        accuracy = numerator / substitute;
    }
    return result;
}
}

```

Написанная функция использует, согласно варианту, разложение в степенной ряд (ряд Тейлора) для нахождения значений.

Для тестирования данной функции был выбран промежуток от -1 до 1, а также были проверены граничные значения и случаи, когда функция должна выкидывать ошибку.

Ниже приведён пример тестов:

```

package task1;

import com.anvisero.task1.ArcTg;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;

import static org.junit.jupiter.api.Assertions.assertAll;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class ArcTgTest {

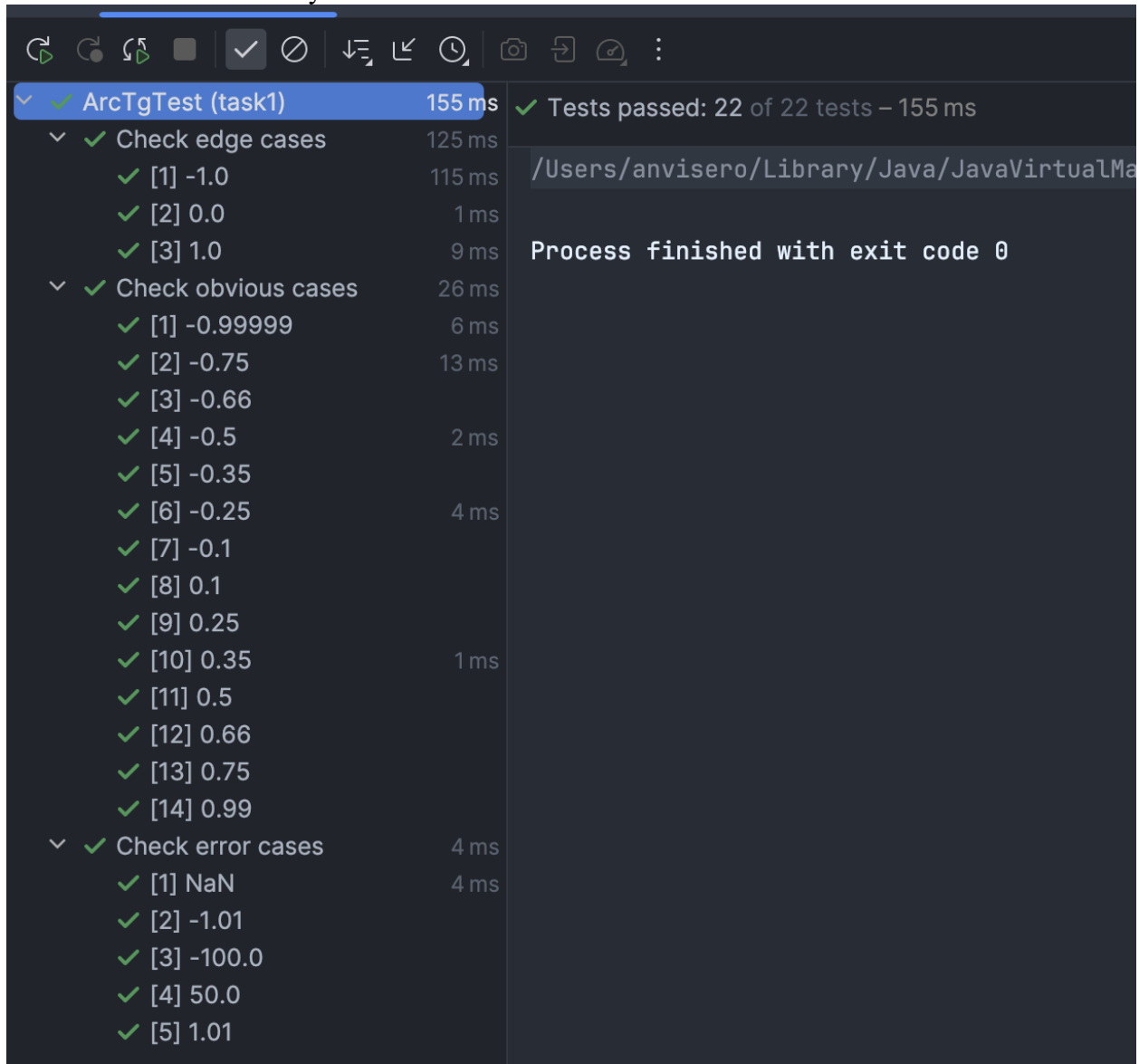
    @ParameterizedTest
    @DisplayName("Check obvious cases")
    @ValueSource(doubles = {-0.99999, -0.75, -0.66, -0.5, -0.35, -0.25, -0.1, 0.1, 0.25, 0.35, 0.5, 0.66, 0.75, 0.99})
    void testCalculateWithObviousCasesThenReturnResult(double param) {
        assertAll(() -> assertEquals(Math.atan(param),
ArcTg.calculate(param), 0.00001,
            "Error occurred while checking argument: " + param));
    }

    @ParameterizedTest
    @DisplayName("Check edge cases")
    @ValueSource(doubles = {-1, 0, 1})
    void testCalculateWithEdgeCasesThenReturnResult(double param) {
        assertAll(() -> assertEquals(Math.atan(param),
ArcTg.calculate(param), 0.00001,
            "Error occurred while checking argument: " + param));
    }

    @ParameterizedTest
    @DisplayName("Check error cases")
    @ValueSource(doubles = {Double.NaN, -1.01, -100, 50, 1.01})
    void testCalculateWithErrorCasesThenReturnResult(double param) {
        assertAll(() -> assertEquals(Double.NaN, ArcTg.calculate(param),
            "Error occurred while checking argument: " + param));
    }
}

```

Все тесты выполнены успешно:



Test Category	Test Case	Duration
Check edge cases	[1] -1.0	115 ms
	[2] 0.0	1 ms
	[3] 1.0	9 ms
Check obvious cases	[1] -0.99999	6 ms
	[2] -0.75	13 ms
	[3] -0.66	
	[4] -0.5	2 ms
	[5] -0.35	
	[6] -0.25	4 ms
	[7] -0.1	
	[8] 0.1	
	[9] 0.25	
	[10] 0.35	1 ms
	[11] 0.5	
	[12] 0.66	
	[13] 0.75	
	[14] 0.99	
Check error cases	[1] NaN	4 ms
	[2] -1.01	4 ms
	[3] -100.0	
	[4] 50.0	
	[5] 1.01	

✓ Tests passed: 22 of 22 tests – 155 ms

/Users/anvisero/Library/Java/JavaVirtualMa

Process finished with exit code 0

Реализация алгоритма QuickSort

```
package com.anvisero.task2;

public class QuickSort<T extends Comparable<T>> {

    public static <T extends Comparable<T>> void sort(T[] array, int low, int
high) {
        if (array == null || low < 0 || high < 0 || low > high || (low >=
array.length || high >= array.length)
            && array.length != 0) {
            throw new IllegalArgumentException();
        }

        quickSort(array, low, high);
    }
}
```

```

    }

    private static <T extends Comparable<T>> void quickSort(T[] array, int
low, int high) {
        if (array.length == 0) {
            return;
        }

        if (low < high) {
            int pivot = partition(array, low, high);

            quickSort(array, low, pivot - 1);
            quickSort(array, pivot + 1, high);
        }
    }

    private static <T extends Comparable<T>> int partition(T[] array, int
low, int high) {
        int middle = low + (high - low) / 2;
        T pivot = array[middle];

        T temp = array[middle];
        array[middle] = array[high];
        array[high] = temp;

        int i = (low - 1);
        for (int j = low; j < high; j++) {
            if (array[j].compareTo(pivot) < 0) {
                i++;

                temp = array[i];
                array[i] = array[j];
                array[j] = temp;
            }
        }

        temp = array[i + 1];
        array[i + 1] = array[high];
        array[high] = temp;

        return i + 1;
    }
}

```

Для тестирования данного алгоритма были выбраны как очевидные случаи, так и сценарии, в которых алгоритм должен выбрасывать ошибку. Более того было проверено неправильное указание границ интервалов сортировки, сортировка пустого массива.

Все тесты выполнены успешно:

```
✓ QuickSortTest (task2) 156 ms
  > ✓ Check sorting all array 108 ms
  > ✓ Check sorting of empty array 6 ms
    ✓ Check null input 19 ms
  > ✓ Check sorting part of array 9 ms
  > ✓ Check sorting of array with ir 14 ms

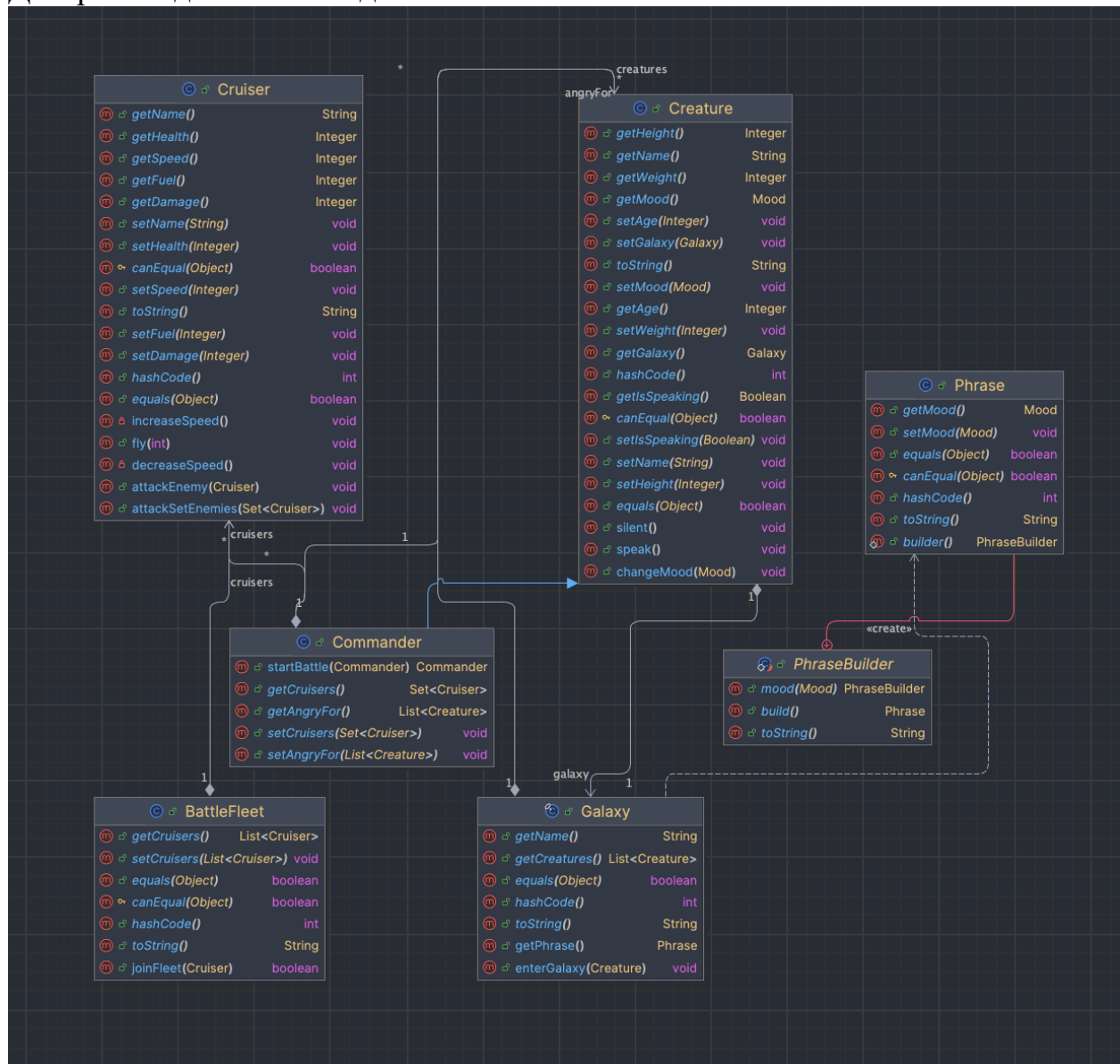
Tests passed: 49 of 49 tests – 156 ms

/Users/anvisero/Library/Java/JavaVirt

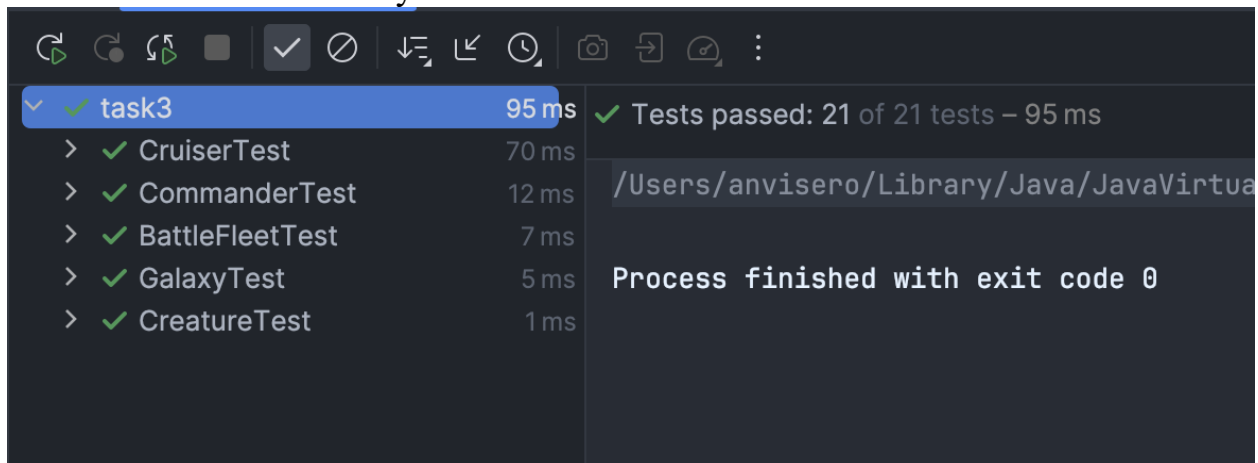
Process finished with exit code 0
```

Доменная модель для заданного текста

Диаграмма доменной модели:



Все тесты выполнены успешно:



Вывод

В процессе выполнения лабораторной работы я научился писать юнит-тесты для разработанных классов и методов. Основная сложность данной работы заключалась в необходимости проявить гибкость мышления при проверке ожидаемого поведения, т. е. придумывать альтернативные способы достижения результата, либо вручную формировать как исходные, так и ожидаемые данные для сравнения. Важно отметить, что достижение 100%-го покрытия очень сложно, поэтому необходимо проверять лишь «избранные» входные данные и использовать анализ эквивалентности.