**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: P33101

Преподаватель: Машина Екатерина Алексеевна



Санкт-Петербург, 2024

**Задание**

A screenshot of a computer

Description automatically generated

**Выполнение**

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

**A graph with a red line

Description automatically generated**

**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 выполнились успешно:  
A screenshot of a computer

Description automatically generated

**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 выполнились успешно:  
A screenshot of a computer

Description automatically generated

**Доменная модель для заданного текста**

Диаграмма доменной модели:  
A screenshot of a computer

Description automatically generated

Все тесты выполнились успешно:  
A screenshot of a computer

Description automatically generated

**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.