**Лабораторна робота №2**

**Основи TDD та JUnit. Основні алгоритмічні структури мови Java**

**Ціль роботи**: оволодіння основами методології **TDD** та навичками роботи з алгоритмічними структурами мови **Java.**

**Завдання**

1. У головному класі описати метод, що обчислює значення функції, яка задана у

таблиці і у тестовому класі - тестові методи для нього.

2. Розробити метод, що за вказаними значеннями кроку, початку та кінця інтервалу

обчислює кількість кроків для табулювання та тестові методи для.

3. Створити методи, що створюють масиви значень функції (y) та її аргументу (x) в

усіх точках вказаного інтервалу із заданим кроком.

4. Створити методи, які після формування масивів, знаходять номери найбільшого

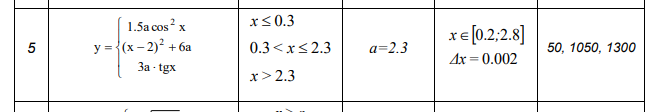
та найменшого елементів масиву значень функції, та методи, що обчислюють та

суму та середнє арифметичне елементів масиву значень функції. Методи

створювати разом з тестами.

5. Створити методи виведення найбільшого та найменшого елементів масиву

значень функції, вказавши їхні номери і відповідні значення аргументу.



**Короткі теоретичні відомості**

*Управляючі структури мови Java:*

1. switch.

Конструкція switch у Java як і у С++ дозволяє передавати управління тому чи

іншому блоку коду, що позначений іменованою міткою, в залежності від

значення виразу.

2. for (each)

Починаючи з версії Java 5 у мові Java з’явилась нова конструкція, призначена

для виконання ітерації по масиву або колекції. Вона виглядає так:

for (<тип елементу> <формальне ім’я> : <масив>) Інструкція

3. Мітки

Інструкції програми можуть бути позначені мітками (labels). Мітка являє

собою змістовне ім’я, що дозволяє посилатися на відповідну інструкцію:

Мітка: Інструкція

Звертатися до мітки дозволено тільки за допомогою команд break та continue

(вони розглядатимуться далі).

4. break

Інструкція break застосовується для завершення виконання коду будь-якого

блоку. Існують дві форми інструкції – безіменна:

break;

та іменована:

break мітка;

Безіменна команда break перериває виконання коду конструкцій switch, for,

while або do і може використовуватися лише всередині цих конструкцій.

Команда break у іменованій формі може перервати виконання будь-якої

інструкції, що помічена відповідною міткою.

*Розв’язання*

1. **Текст програми**

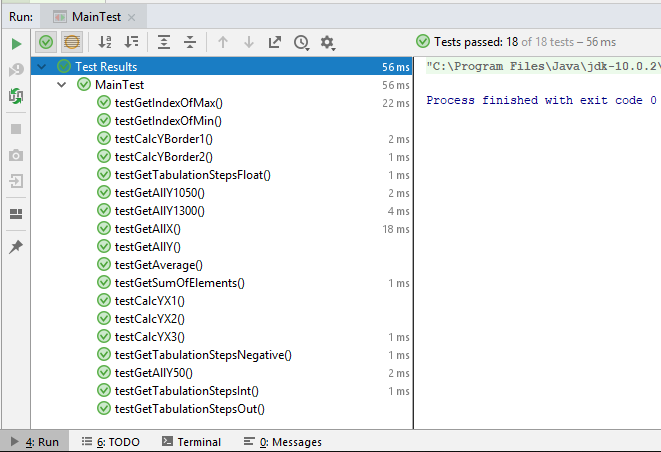
Main.java:

**package** com.company;  
  
**import** java.util.Scanner;  
**import static** java.lang.Math.\*;  
  
**public class** Main {  
 **private double** askDouble(String title) {  
 Scanner scanner = **new** Scanner(System.***in***);  
  
 System.***out***.println(title);  
 **return** scanner.nextDouble();  
 }  
  
 **public void** printEl(**double**[] arr, **int** index) {  
 System.***out***.println(**"arr["** + index + **"]"** + **" = "** + arr[index]);  
 }  
  
 **public double** getAverage(**double**[] arr) {  
 **return** getSumOfElements(arr) / arr.**length**;  
 }  
  
 **public double** getSumOfElements(**double**[] arr) {  
 **double** res = 0;  
  
 **for** (**double** item : arr) {  
 res += item;  
 }  
  
 **return** res;  
 }  
  
 **public int** getIndexOfMin(**double**[] arr) {  
 **double** min = arr[0];  
 **int** index = 0;  
  
 **for** (**int** i = 1; i < arr.**length**; i++) {  
 **if** (arr[i] < min) {  
 min = arr[i];  
 index = i;  
 }  
 }  
  
 **return** index;  
 }  
  
 **public int** getIndexOfMax(**double**[] arr) {  
 **double** max = arr[0];  
 **int** index = 0;  
  
 **for** (**int** i = 1; i < arr.**length**; i++) {  
 **if** (arr[i] > max) {  
 max = arr[i];  
 index = i;  
 }  
 }  
  
 **return** index;  
 }  
  
 **public double**[] getAllX(**double** start, **double** end, **double** step) {  
 **int** steps = getTabulationStepsCount(start, end, step);  
 **double**[] allX = **new double**[steps];  
  
 **for** (**int** i = 0; i < steps; i++) {  
 **double** res = start + i \* step;  
 allX[i] = *round*(res \* 10000.0) / 10000.0;  
 }  
  
 **return** allX;  
 }  
  
 **public double**[] getAllY(**double**[] allX, **double** a) {  
 **double**[] allY = **new double**[allX.**length**];  
  
 **for** (**int** i = allX.**length** - 1; i >= 0; i--) {  
 allY[i] = calcY(allX[i], a);  
 }  
  
 **return** allY;  
 }  
  
 **public int** getTabulationStepsCount(**double** start, **double** end, **double** step) {  
 **return** (**int**)((end - start) / step + 0.000001) + 1;  
 }  
  
 **public double** calcY(**double** x, **double** a, **double** epsilon) {  
 **if** (x - 0.3 <= epsilon) {  
 **return** 1.5 \* a \* *cos*(x) \* *cos*(x);  
  
 } **else if** (x - 2.3 <= epsilon) {  
 **return** (x - 2) \* (x - 2) + (6 \* a);  
  
 } **else** {  
 **return** 3 \* a \* *tan*(x);  
 }  
 }  
  
 **public double** calcY(**double** x, **double** a) {  
 **return** calcY(x, a, 0.001);  
 }  
  
}

MainTest.java

**package** com.company;  
  
**import** org.junit.jupiter.api.BeforeEach;  
**import** org.junit.jupiter.api.Test;  
  
**import static** org.junit.jupiter.api.Assertions.\*;  
  
**class** MainTest {  
 Main **main**;  
  
 @BeforeEach  
 **void** setUp() {  
 **main** = **new** Main();  
 }  
  
 *// - - - CalcY - - -* @Test  
 **void** testCalcYX1() {  
 **double** expected = 7.20397873;  
 **double** result = **main**.calcY(0.2, 5);  
 *assertEquals*(expected, result, 0.001, **"По первому уравнению: caclY(0.2, 5)"**);  
 }  
  
 @Test  
 **void** testCalcYX2() {  
 **double** expected = -27.44;  
 **double** result = **main**.calcY(0.4, -5);  
 *assertEquals*(expected, result, 0.001, **"По второму уравнению: caclY(0.4, -5)"**);  
 }  
  
 @Test  
 **void** testCalcYX3() {  
 **double** expected = -0.855279258;  
 **double** result = **main**.calcY(3, 2);  
 *assertEquals*(expected, result, 0.001, **"По третьему уравнению: caclY(0.4, -5)"**);  
 }  
  
 @Test  
 **void** testCalcYBorder1() {  
 **double** expected = 1.369;  
 **double** result = **main**.calcY(0.3000000000001, 1);  
 *assertEquals*(expected, result, 0.001, **"x == 0.3000..001 должен считаться по первому уравнению"**);  
 }  
  
 @Test  
 **void** testCalcYBorder2() {  
 **double** expected = 6.09;  
 **double** result = **main**.calcY(2.300000000001, 1);  
 *assertEquals*(expected, result, 0.001, **"x == 2.300..01 должен считаться по второму уравнению"**);  
 }  
  
 *// - - - TabulationSteps - - -* @Test  
 **void** testGetTabulationStepsInt() {  
 **double** expected = 10;  
 **double** result = **main**.getTabulationStepsCount(1, 10, 1);  
 *assertEquals*(expected, result, 0.001, **"start: 1, end: 10, step: 1 -> 10"**);  
 }  
  
 @Test  
 **void** testGetTabulationStepsFloat() {  
 **double** expected = 11;  
 **double** result = **main**.getTabulationStepsCount(0.1, 1.1, 0.1);  
 *assertEquals*(expected, result, 0.001, **"start: 0.1, end: 1.1, step: 0.1 -> 11"**);  
 }  
  
 @Test  
 **void** testGetTabulationStepsOut() {  
 **double** expected = 6;  
 **double** result = **main**.getTabulationStepsCount(1, 12, 2);  
 *assertEquals*(expected, result, 0.001, **"start: 1, end: 12, step: 2 -> 6"**);  
 }  
  
 @Test  
 **void** testGetTabulationStepsNegative() {  
 **double** expected = 5;  
 **double** result = **main**.getTabulationStepsCount(-10, -1, 2);  
 *assertEquals*(expected, result, 0.001, **"start: -10, end: -1, step: 2 -> 5"**);  
 }  
  
 *// - - - getX - - -* @Test  
 **void** testGetAllX() {  
 **double**[] expected = {1, 2, 3, 4, 5};  
 **double**[] result = **main**.getAllX(1, 5, 1);  
 *assertArrayEquals*(expected, result, 0.001, **"getAllX(1, 5, 1) -> {1,2,3,4,5}"**);  
 }  
  
 @Test  
 **void** testGetAllY() {  
 **double** a = 2.3;  
 **double**[] allX = {1, 2};  
  
 **double**[] expected = {**main**.calcY(allX[0], a), **main**.calcY(allX[1], a)};  
 **double**[] result = **main**.getAllY(allX, a);  
 *assertArrayEquals*(expected, result, 0.001, **"getAllY({1,2}, 2.3)"**);  
 }  
  
 *// - - - Tabulation - - -* @Test  
 **void** testGetAllY50() {  
 **double** a = 2.3, start = 0.2, end = 2.8, step = 0.002;  
 **double**[] allX = **main**.getAllX(start, end, step);  
 **double**[] allY = **main**.getAllY(allX, a);  
  
 **double** expected = 3.1487; *//x = 0.3* **double** result = allY[50];  
 *assertEquals*(expected, result, 0.001, **"allY[50]"**);  
 }  
  
 @Test  
 **void** testGetAllY1050() {  
 **double** a = 2.3, start = 0.2, end = 2.8, step = 0.002;  
 **double**[] allX = **main**.getAllX(start, end, step);  
 **double**[] allY = **main**.getAllY(allX, a);  
  
 **double** expected = 13.89;  
 **double** result = allY[1050];  
 *assertEquals*(expected, result, 0.001, **"allY[1050]"**);  
 }  
  
 @Test  
 **void** testGetAllY1300() {  
 **double** a = 2.3, start = 0.2, end = 2.8, step = 0.002;  
 **double**[] allX = **main**.getAllX(start, end, step);  
 **double**[] allY = **main**.getAllY(allX, a);  
  
 **double** expected = -2.4532;  
 **double** result = allY[1300];  
 *assertEquals*(expected, result, 0.001, **"allY[1300]"**);  
 }  
  
 *// - - - getMinMax - - -* @Test  
 **void** testGetIndexOfMin() {  
 **double**[] arr = {1, -2, 2};  
  
 **double** expected = 1;  
 **double** result = **main**.getIndexOfMin(arr);  
 *assertEquals*(expected, result, 0.001, **"getIndexOfMin({1,-2,2}) -> 1"**);  
 }  
  
 @Test  
 **void** testGetIndexOfMax() {  
 **double**[] arr = {1, 2, -2};  
  
 **double** expected = 1;  
 **double** result = **main**.getIndexOfMax(arr);  
 *assertEquals*(expected, result, 0.001, **"getIndexOfMax({1,2,-2}) -> 1"**);  
 }  
  
 *// - - - Average - - -* @Test  
 **void** testGetSumOfElements() {  
 **double**[] arr = {1,2,3,4};  
 **double** expected = 10;  
 **double** result = **main**.getSumOfElements(arr);  
 *assertEquals*(expected, result, 0.001, **"getSumOfElements({1,2,3,4}) -> 10"**);  
 }  
  
 @Test  
 **void** testGetAverage() {  
 **double**[] arr = {1,2,3,4};  
 **double** expected = 2.5;  
 **double** result = **main**.getAverage(arr);  
 *assertEquals*(expected, result, 0.001, **"getAverage({1,2,3,4}) -> 2.5"**);  
 }  
}

1. Результат виконання програми:



**Висновок:** на цій лабораторній роботі оволодів основами методології **TDD** та навичками роботи з алгоритмічними структурами мови **Java.**