Университет ИТМО

Кафедра ВТ

Лабораторная работа №1

по дисциплине Тестирование ПО

***Вариант 686***

Выполнили: Толстухин Максим

Татаринов Данил

Группа: P3411

Преподаватель: Клеменков Сергей Викторович

Санкт-Петербург

2019

Задание  
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

1. Функция sin(x)
2. Программный модуль для работы с левосторонней кучей
3. Описание предметной области:

Форд подскочил к пультам; он смог догадаться о назначении некоторых рукояток и схватился за них. Корабль затрясло оттого, что его двигатели с визгом начали толкать его одновременно во всех направлениях. Форд отпустил половину рукояток и корабль, описав крутую дугу, развернулся кругом и направился навстречу ракетам.

Пункт1

Функция Sin(x) эквивалентна на промежутке [-pi/2 \* k; pi/2 \* k]

Граничные значения: 0, +-pi/2

Тестирование было проведено для краевых точек функции и нескольких точек между краевыми.

Проведены тестирования на значениях: 0, +-pi/2, +-5pi/6, +-2pi/3, +-pi/3, +-pi/6  
Для каждого из значений проведены тесты со смещением кратным pi = +-6pi

Код тестов:

@Test

public void expect\_null\_sin\_of\_null()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0, sinus.sin(0), 0.001);

}

@Test

public void expect\_null\_sin\_of\_six\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0, sinus.sin(Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_null\_sin\_of\_minus\_six\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0, sinus.sin(-Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_null\_sin\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0, sinus.sin(Sinus.PI), 0.001);

}

@Test

public void expect\_null\_sin\_of\_pi\_plus\_six\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0, sinus.sin(Sinus.PI + Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_null\_sin\_of\_pi\_minus\_six\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0, sinus.sin(Sinus.PI - Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_one\_sin\_of\_half\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(1, sinus.sin(Sinus.PI / 2), 0.001);

}

@Test

public void expect\_one\_sin\_of\_half\_plus\_six\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(1, sinus.sin(Sinus.PI / 2 + Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_one\_sin\_of\_half\_minus\_six\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(1, sinus.sin(Sinus.PI / 2 - Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_one\_sin\_of\_minus\_half\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(-1, sinus.sin(-Sinus.PI / 2), 0.001);

}

@Test

public void expect\_one\_sin\_of\_minus\_half\_plus\_six\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(-1, sinus.sin(-Sinus.PI / 2 + Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_one\_sin\_of\_minus\_half\_minus\_six\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(-1, sinus.sin(-Sinus.PI / 2 - Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_sqrt\_of\_three\_divide\_by\_two\_sin\_of\_one\_divide\_by\_three\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0.86602540378443864676372317075294, sinus.sin(Sinus.PI / 3), 0.001);

}

@Test

public void expect\_sqrt\_of\_three\_divide\_by\_two\_sin\_of\_one\_divide\_by\_three\_plus\_six\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0.86602540378443864676372317075294, sinus.sin(Sinus.PI / 3 + Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_sqrt\_of\_three\_divide\_by\_two\_sin\_of\_one\_divide\_by\_three\_minus\_six\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0.86602540378443864676372317075294, sinus.sin(Sinus.PI / 3 - Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_minus\_sqrt\_of\_three\_divide\_by\_two\_sin\_of\_minus\_one\_divide\_by\_three\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(-0.86602540378443864676372317075294, sinus.sin(-Sinus.PI / 3), 0.001);

}

@Test

public void expect\_minus\_sqrt\_of\_three\_divide\_by\_two\_sin\_of\_minus\_one\_divide\_by\_three\_plus\_six\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(-0.86602540378443864676372317075294, sinus.sin(-Sinus.PI / 3 + Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_minus\_sqrt\_of\_three\_divide\_by\_two\_sin\_of\_minus\_one\_divide\_by\_three\_minus\_six\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(-0.86602540378443864676372317075294, sinus.sin(-Sinus.PI / 3 - Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_half\_sin\_of\_on\_divide\_by\_six\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0.5, sinus.sin(Sinus.PI / 6), 0.001);

}

@Test

public void expect\_half\_sin\_of\_on\_divide\_by\_six\_plus\_six\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0.5, sinus.sin(Sinus.PI / 6 + Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_half\_sin\_of\_on\_divide\_by\_six\_minus\_six\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0.5, sinus.sin(Sinus.PI / 6 - Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_sqrt\_of\_three\_divide\_by\_two\_sin\_of\_pi\_minus\_one\_divide\_by\_three\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0.86602540378443864676372317075294, sinus.sin(Sinus.PI - Sinus.PI / 3), 0.001);

}

@Test

public void expect\_sqrt\_of\_three\_divide\_by\_two\_sin\_of\_pi\_minus\_one\_divide\_by\_three\_of\_pi\_plus\_six\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0.86602540378443864676372317075294, sinus.sin(Sinus.PI - Sinus.PI / 3 + Sinus.PI \*2), 0.001);

}

@Test

public void expect\_sqrt\_of\_three\_divide\_by\_two\_sin\_of\_pi\_minus\_one\_divide\_by\_three\_of\_pi\_minus\_six\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0.86602540378443864676372317075294, sinus.sin(Sinus.PI - Sinus.PI / 3 - Sinus.PI \*2), 0.001);

}

@Test

public void expect\_minus\_sqrt\_of\_three\_divide\_by\_two\_sin\_of\_minus\_pi\_plus\_one\_divide\_by\_three\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(-0.86602540378443864676372317075294, sinus.sin(-Sinus.PI + Sinus.PI / 3), 0.001);

}

@Test

public void expect\_minus\_sqrt\_of\_three\_divide\_by\_two\_sin\_of\_minus\_pi\_plus\_one\_divide\_by\_three\_of\_pi\_plus\_six\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(-0.86602540378443864676372317075294, sinus.sin(-Sinus.PI + Sinus.PI / 3 + Sinus.PI \*2), 0.001);

}

@Test

public void expect\_minus\_sqrt\_of\_three\_divide\_by\_two\_sin\_of\_minus\_pi\_plus\_one\_divide\_by\_three\_of\_pi\_minus\_six\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(-0.86602540378443864676372317075294, sinus.sin(-Sinus.PI + Sinus.PI / 3 - Sinus.PI \*2), 0.001);

}

@Test

public void expect\_minus\_half\_sin\_of\_minus\_pi\_divide\_by\_six()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(-0.5, sinus.sin(-Sinus.PI / 6), 0.001);

}

@Test

public void expect\_minus\_half\_sin\_of\_minus\_pi\_divide\_by\_six\_plus\_six\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(-0.5, sinus.sin(-Sinus.PI / 6 + Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_minus\_half\_sin\_of\_minus\_pi\_divide\_by\_six\_minus\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(-0.5, sinus.sin(-Sinus.PI / 6 - Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_half\_sin\_of\_pi\_minus\_one\_divide\_by\_six\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0.5, sinus.sin(Sinus.PI - Sinus.PI / 6), 0.001);

}

@Test

public void expect\_half\_sin\_of\_pi\_minus\_one\_divide\_by\_six\_plus\_six\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0.5, sinus.sin(Sinus.PI - Sinus.PI / 6 + Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_half\_sin\_of\_pi\_minus\_one\_divide\_by\_six\_minus\_six\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(0.5, sinus.sin(Sinus.PI - Sinus.PI / 6 - Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_minus\_half\_sin\_of\_minus\_pi\_plus\_one\_divide\_by\_six\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(-0.5, sinus.sin(-Sinus.PI + Sinus.PI / 6), 0.001);

}

@Test

public void expect\_minus\_\_half\_sin\_of\_minus\_pi\_plus\_one\_divide\_by\_six\_plus\_six\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(-0.5, sinus.sin(-Sinus.PI + Sinus.PI / 6 + Sinus.PI \* 6), 0.001);

}

@Test

public void expect\_minus\_half\_sin\_of\_minus\_pi\_plus\_one\_divide\_by\_six\_minus\_six\_of\_pi()

{

Sinus sinus = new Sinus();

Assertions.assertEquals(-0.5, sinus.sin(-Sinus.PI + Sinus.PI / 6 - Sinus.PI \* 6), 0.001);

}

Код функции:

public class Sinus {

public static double PI = 3.1415926535897932384626433832795;

public int factorial(int index) {

if (index < 0) {

throw new NegativePowException();

}

int multiply = 1;

for (int i = 1; i <= index; i++) {

multiply \*= i;

}

return multiply;

}

public double element(double x, int power) {

return pow(-1, power) \* pow(x, 2 \* power + 1) / factorial(2 \* power + 1);

}

public double sin(double x) {

x = x % 6.283185307179586476925286766559;

if (x < 0) {

x = 6.283185307179586476925286766559 + x;

}

if (x >= 3.1415926535897932384626433832795) {

x = x - 6.283185307179586476925286766559;

}

if (x > 1.5707963267948966192313216916398) {

x = 3.1415926535897932384626433832795 - x;

}

if (x < -1.5707963267948966192313216916398) {

x = -3.1415926535897932384626433832795 - x;

}

double sum = 0;

for (int i = 0; i < 15; i++) {

sum += element(x, i);

}

return sum;

}

public double pow(double x, int power) {

if (power < 0) {

throw new NegativePowException();

}

double multiply = 1;

for (int i = 0; i < power; i++) {

multiply \*= x;

}

return multiply;

}

}

Пункт2

Код алгоритма:

/\*\*

\* Java Program to Implement LeftistHeap

\*\*/

/\*\* Class LeftHeapNode \*\*/

class LeftHeapNode

{

int element, sValue;

LeftHeapNode left, right;

public LeftHeapNode(int ele)

{

this(ele, null, null);

}

public LeftHeapNode(int ele, LeftHeapNode left, LeftHeapNode right)

{

this.element = ele;

this.left = left;

this.right = right;

this.sValue = 0;

}

}

/\*\* Class LeftistHeap \*\*/

class LeftistHeap

{

private LeftHeapNode root;

/\*\* Constructor \*\*/

public LeftistHeap()

{

root = null;

}

/\*\* Check if heap is empty \*\*/

public boolean isEmpty()

{

return root == null;

}

/\*\* Make heap empty \*\*/

public void clear( )

{

root = null;

}

/\*\* Function to insert data \*\*/

public void insert(int x )

{

root = merge(new LeftHeapNode( x ), root);

}

/\*\* Function merge \*\*/

private LeftHeapNode merge(LeftHeapNode x, LeftHeapNode y)

{

if (x == null)

return y;

if (y == null)

return x;

if (x.element > y.element)

{

LeftHeapNode temp = x;

x = y;

y = temp;

}

x.right = merge(x.right, y);

if(x.left == null)

{

x.left = x.right;

x.right = null;

}

else

{

if(x.left.sValue < x.right.sValue)

{

LeftHeapNode temp = x.left;

x.left = x.right;

x.right = temp;

}

x.sValue = x.right.sValue + 1;

}

return x;

}

/\*\* Function to delete minimum element \*\*/

public int deleteMin( )

{

if (isEmpty() )

return -1;

int minItem = root.element;

root = merge(root.left, root.right);

return minItem;

}

/\*\* Inorder traversal \*\*/

public String inorder()

{

return inorder(root);

}

private String inorder(LeftHeapNode r)

{

if (r != null)

{

String string = "";

string = string.concat(inorder(r.left));

string = string.concat(r.element +" ");

string = string.concat(inorder(r.right));

return string;

}

return "";

}

}

Код тестов:

@Provide

Arbitrary<List<Integer>> first(){

return Arbitraries.integers().list();

}

@Property

void listheap\_isnt\_empty(@ForAll("first")List<Integer> any\_ints){

LeftistHeap lh = new LeftistHeap();

for(int i : any\_ints){

lh.insert(i);

}

lh.insert(0);

assertThat(lh.isEmpty()).isFalse();

}

@Property

void listheap\_is\_empty(@ForAll("first")List<Integer> any\_ints){

LeftistHeap lh = new LeftistHeap();

for(int i : any\_ints){

lh.insert(i);

}

lh.clear();

assertThat(lh.isEmpty()).isTrue();

}

@Property

void listheap\_is\_empty\_after\_manual\_remove(@ForAll("first")List<Integer> any\_ints){

LeftistHeap lh = new LeftistHeap();

for(int i : any\_ints){

lh.insert(i);

}

for(int i = 0; i < any\_ints.size(); i++){

lh.deleteMin();

}

assertThat(lh.isEmpty()).isTrue();

}

@Example

void listheap\_delete\_empty(){

LeftistHeap lh = new LeftistHeap();

assertThat(lh.deleteMin() == -1).isTrue();

}

@Example

void listheap\_simple\_pair\_insert(){

LeftistHeap lh = new LeftistHeap();

lh.insert(1);

lh.insert(0);

assertThat(lh.inorder()).isEqualTo("1 0 ");

}

@Property

void listheap\_sort(@ForAll("first")List<Integer> any\_ints){

LeftistHeap lh = new LeftistHeap();

for(int i : any\_ints){

lh.insert(i);

}

any\_ints.sort(Comparator.naturalOrder());

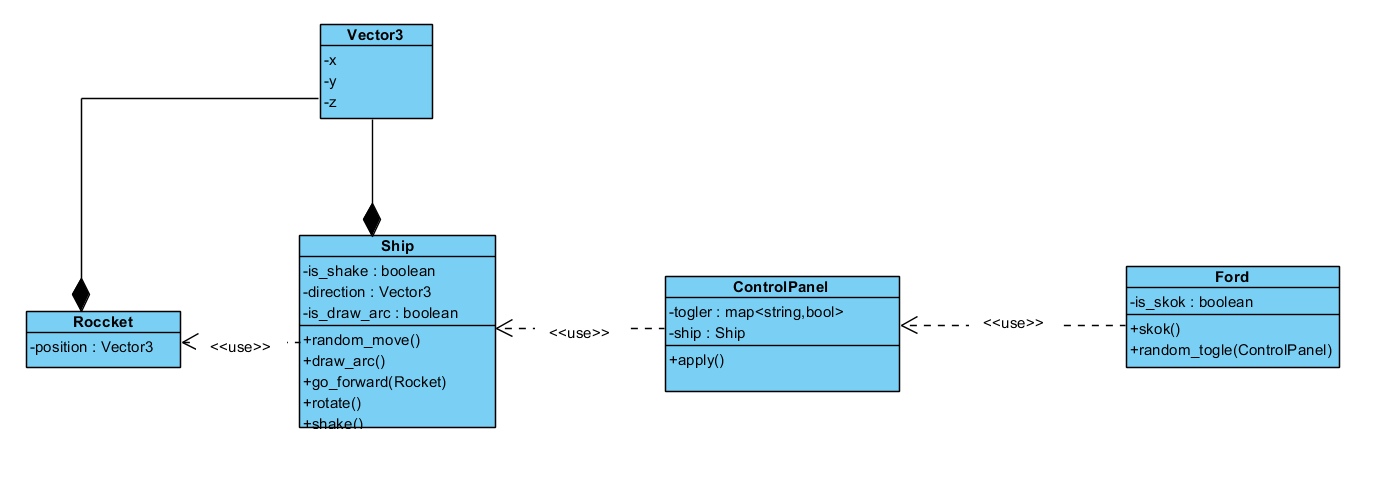
for (int i = 0; i < any\_ints.size(); i++) {

assertThat(lh.deleteMin()).isEqualTo(any\_ints.get(i));

}

}

Пункт3:

Код тестов:

public class ControllPanaelTest {

@Provide

Arbitrary<Map<String, Boolean>> first() {

Arbitrary<String> keys = Arbitraries.strings()

.withSamples("start")

.withSamples("draw arc")

.withSamples("rotate");

Arbitrary<Boolean> values = Arbitraries.constant(false);

return Arbitraries.maps(keys, values);

}

@Property

boolean disabled\_toglers(@ForAll("first") HashMap<String, Boolean> disabled\_toglers)

{

Ship ship = new Ship();

ControlPanel controlPanel = new ControlPanel();

controlPanel.ship = ship;

controlPanel.toglers = disabled\_toglers;

controlPanel.Apply();

return !ship.isShaking && !ship.hasDrawnArc && !ship.hasRoteted && ship.direction == null;

}

@Example

void empty\_togler\_test()

{

Ship ship = new Ship();

ControlPanel controlPanel = new ControlPanel();

controlPanel.ship = ship;

controlPanel.Apply();

assertThat(ship.isShaking).isFalse();

assertThat(ship.hasDrawnArc).isFalse();

assertThat(ship.hasRoteted).isFalse();

assertThat(ship.direction).isNull();

}

@Example

void start\_togler\_test()

{

Ship ship = new Ship();

ControlPanel controlPanel = new ControlPanel();

controlPanel.ship = ship;

controlPanel.toglers.put("start", true);

controlPanel.Apply();

assertThat(ship.isShaking).isTrue();

assertThat(ship.hasDrawnArc).isFalse();

assertThat(ship.hasRoteted).isFalse();

assertThat(ship.direction).isNull();

}

@Example

void draw\_arc\_togler\_test()

{

Ship ship = new Ship();

ControlPanel controlPanel = new ControlPanel();

controlPanel.ship = ship;

controlPanel.toglers.put("draw arc", true);

controlPanel.Apply();

assertThat(ship.isShaking).isFalse();

assertThat(ship.hasDrawnArc).isTrue();

assertThat(ship.hasRoteted).isFalse();

assertThat(ship.direction).isNull();

}

@Example

void rotate\_togler\_test()

{

Ship ship = new Ship();

ControlPanel controlPanel = new ControlPanel();

controlPanel.ship = ship;

controlPanel.toglers.put("rotate", true);

controlPanel.Apply();

assertThat(ship.isShaking).isFalse();

assertThat(ship.hasDrawnArc).isFalse();

assertThat(ship.hasRoteted).isTrue();

assertThat(ship.direction).isNull();

}

@Example

void rocket\_togler\_test()

{

Ship ship = new Ship();

ControlPanel controlPanel = new ControlPanel();

controlPanel.ship = ship;

Rockets test\_rocket = new Rockets("test rocket");

controlPanel.rockets.add(test\_rocket);

controlPanel.toglers.put("test rocket", true);

controlPanel.Apply();

assertThat(ship.isShaking).isFalse();

assertThat(ship.hasDrawnArc).isFalse();

assertThat(ship.hasRoteted).isFalse();

assertThat(ship.direction).isEqualTo(test\_rocket.position);

}

}

public class FordTest {

@Provide

Arbitrary<Map<String, Boolean>> first() {

Arbitrary<String> keys = Arbitraries.strings();

Arbitrary<Boolean> values = Arbitraries.samples(true, false);

return Arbitraries.maps(keys, values).ofMinSize(1);

}

@Property

boolean didnt\_sckoc(@ForAll("first") HashMap<String, Boolean> any\_toglers){

Ford ford = new Ford(new TrueBoolean());

Ship ship = new Ship();

ControlPanel cp = new ControlPanel();

cp.ship = ship;

cp.toglers = any\_toglers;

ford.RandomTogle(cp);

return !ship.isShaking && !ship.hasDrawnArc && !ship.hasRoteted && ship.direction == null;

}

@Property

void did\_sckoc(@ForAll("first") HashMap<String, Boolean> any\_toglers){

Ford ford = new Ford(new TrueBoolean());

ford.ToSckok();

Ship ship = new Ship();

ControlPanel cp = new ControlPanel();

cp.ship = ship;

any\_toglers.put("start", false);

any\_toglers.put("draw arc", false);

any\_toglers.put("rotate", false);

cp.toglers = any\_toglers;

ford.RandomTogle(cp);

assertThat(ship.isShaking).isTrue();

assertThat(ship.hasDrawnArc).isTrue();

assertThat(ship.hasRoteted).isTrue();

}

}

public class ShipTest {

@Provide

Arbitrary<Rockets> first() {

Arbitrary<Integer> x = Arbitraries.integers();

Arbitrary<Integer> y = Arbitraries.integers();

Arbitrary<Integer> z = Arbitraries.integers();

Arbitrary<Vector3> vector3 = Combinators.combine(x, y, z).as(Vector3::new);

Arbitrary<String> rocket\_name = Arbitraries.strings();

return Combinators.combine(rocket\_name, vector3).as(Rockets::new);

}

@Property

boolean ship\_rocket\_test\_x(@ForAll("first") Rockets rockets)

{

Ship ship = new Ship();

ship.GoToRockets(rockets);

return rockets.position.x == ship.direction.x;

}

@Property

boolean ship\_rocket\_test\_y(@ForAll("first") Rockets rockets)

{

Ship ship = new Ship();

ship.GoToRockets(rockets);

return rockets.position.y == ship.direction.y;

}

@Property

boolean ship\_rocket\_test\_z(@ForAll("first") Rockets rockets)

{

Ship ship = new Ship();

ship.GoToRockets(rockets);

return rockets.position.z == ship.direction.z;

}

}

Вывод:

Было получено представление о юнит тестировании и были получены практические навыки определения классов эквивалентности и написания юнит тестов, были опробованы способы тестирования белым и чёрным ящиком, найден/изучен новый для нас вид тестирования - Property Based Testing, его характеристики, особенности применения на практике.