Федеральное государственное автономное образовательное учреждение высшего образования

Университет ИТМО

Дисциплина: Тестирование программного обеспечения

Лабораторная работа 1

Вариант 100063

Выполнили:
Марков Петр Денисович
Кривоносов Егор Дмитриевич

Группа: P33111

Преподаватель:
Яркеев Александр Сергеевич

2022 г.

Санкт-Петербург

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

Вариант 100063

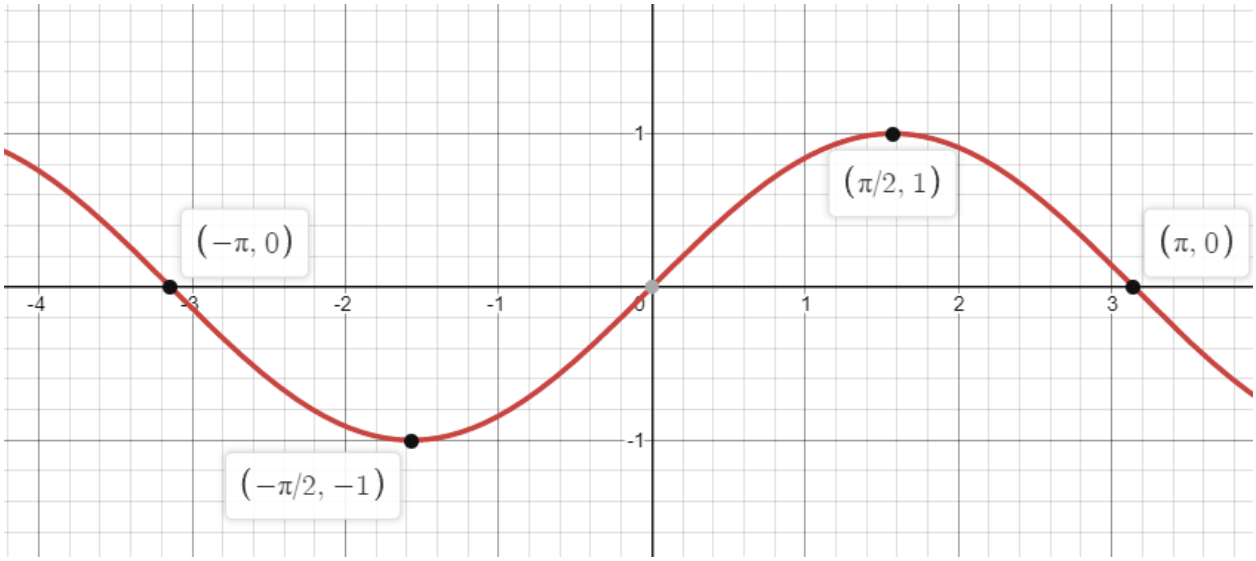
1. Функция sin(x)
2. Программный модуль для пирамидальной сортировки массива (http://www.cs.usfca.edu/~galles/visualization/HeapSort.html)
3. Описание предметной области:

```
Зажужжал мотор.
Тоненький свист перерос в рев воздуха, вырывающегося в черную
пустоту, усеянную невероятно яркими светящимися точками.
Форд и Артур вылетели в открытый космос, как конфетти из хлопушки.
Глава 8
```

Выполнение

Исходный код: https://github.com/RedGry/TPO-LAB-1

sin(x)



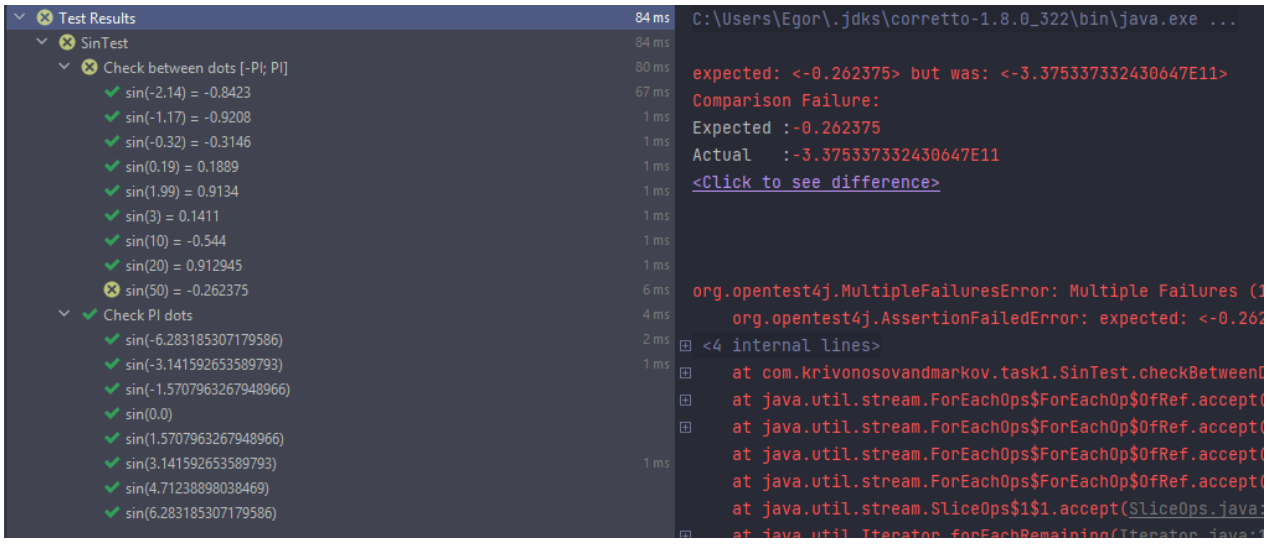
Значения, в которых необходимо проверить функцию: x {-pi/2, 0, pi/2} – граничные значения, где функция меняет знак, а также больших значений x. Для достоверности возьмём также пару значений из промежутка. Итоговая таблица для проверки:

X	-pi	-pi/2	0	pi/2	pi	2pi	1.99	20
Sin(x)	0	-1	0	1	0	0	0.91341	0.91295

```java
public class Sin {

    // https://function-x.ru/chapter9-4/rows4_clip_image063.gif

    public static double calc(double x, int n){
        double result = 0;  // Хранится результат

        double xx = x * x;  // x^2
        double pow = x;     // Для Вычисления числителя
        double fact = 1;    // Для вычисления факториала знаменателя

        int sign = 1;       // Отвечает за знак (чередование '+' -> '-' -> '+' -> ...)

        for (int i = 1; i < n; i += 2){
            fact /= i;
            result += sign * pow * fact;    // (-1)^(n-1) * x^(2n-1) / (2n-1)!
            sign = -sign;
            fact /= (i + 1);
            pow *= xx;
        }

        // Дополнительно округляем, чтобы легче было проверять
        return result;
    }
}
```

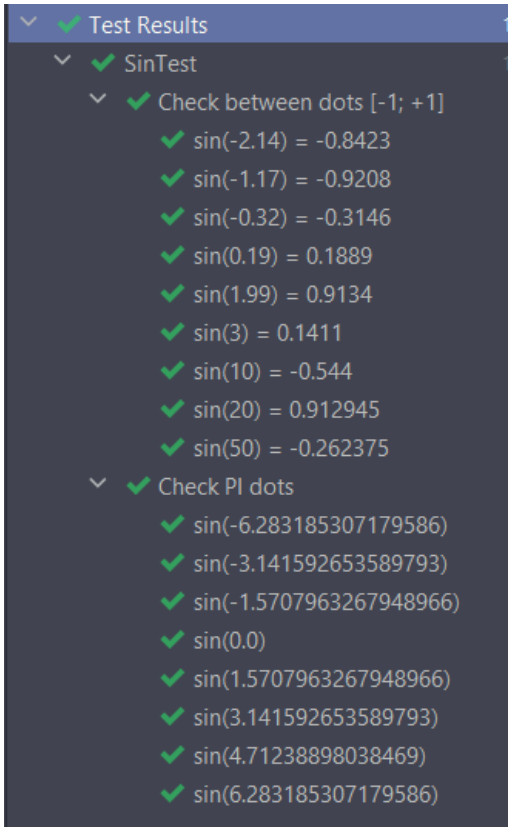
Написанная функция использует, согласно варианту, разложение в степенной ряд (ряд Тейлора) для нахождения значений.

Функция не проходит на больших значениях. Ряд Тейлора (степенной) - плохой способ нахождения , потому что его ошибка увеличивается по мере удаления от аргумента, вокруг которой находится ряд. Поэтому тест с большим достаточно значением радиан не проходит.



Чтобы решить данную проблему можно сделать так, чтобы пока у нас радианы не станут меньше 2 * PI их уменьшаем на 2 * PI тем самым не теряем четность и получаем при больших значениях правильный ответ.

```
double PI2 = Math.PI * 2;

if (x >= 0) {
    while (x > PI2) {
        x -= 2 * PI2;
    }
} else if (x < 0){
    while (x < PI2) {
        x += PI2;
    }
}
```



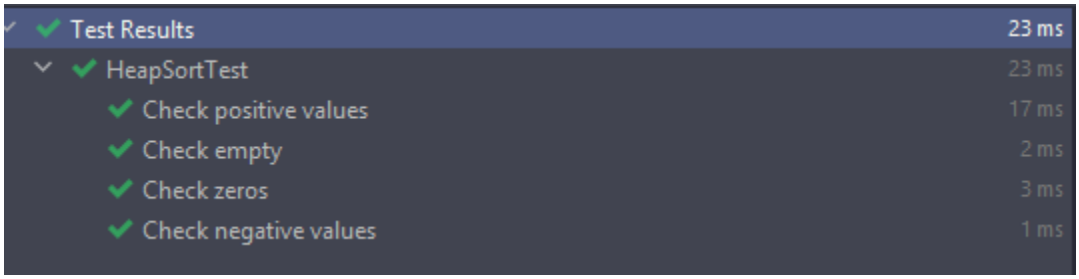
HeapSort

```java
public class HeapSort {
    public static int[] sort(int[] arr)
    {
        int n = arr.length;

        // Build heap (rearrange array)
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

        // One by one extract an element from heap
        for (int i = n - 1; i > 0; i--) {
            // Move current root to end
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            // call max heapify on the reduced heap
            heapify(arr, i,  0);
        }

        return arr;
    }

    // To heapify a subtree rooted with node i which is
    // an index in arr[]. n is size of heap
    static void heapify(int[] arr, int n, int i)
    {
        int largest = i; // Initialize largest as root
        int l = 2 * i + 1; // left = 2*i + 1
        int r = 2 * i + 2; // right = 2*i + 2

        // If left child is larger than root
        if (l < n && arr[l] > arr[largest])
            largest = l;

        // If right child is larger than largest so far
        if (r < n && arr[r] > arr[largest])
            largest = r;

        // If largest is not root
        if (largest != i) {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;

            // Recursively heapify the affected sub-tree
            heapify(arr, n, largest);
        }
    }

}
```

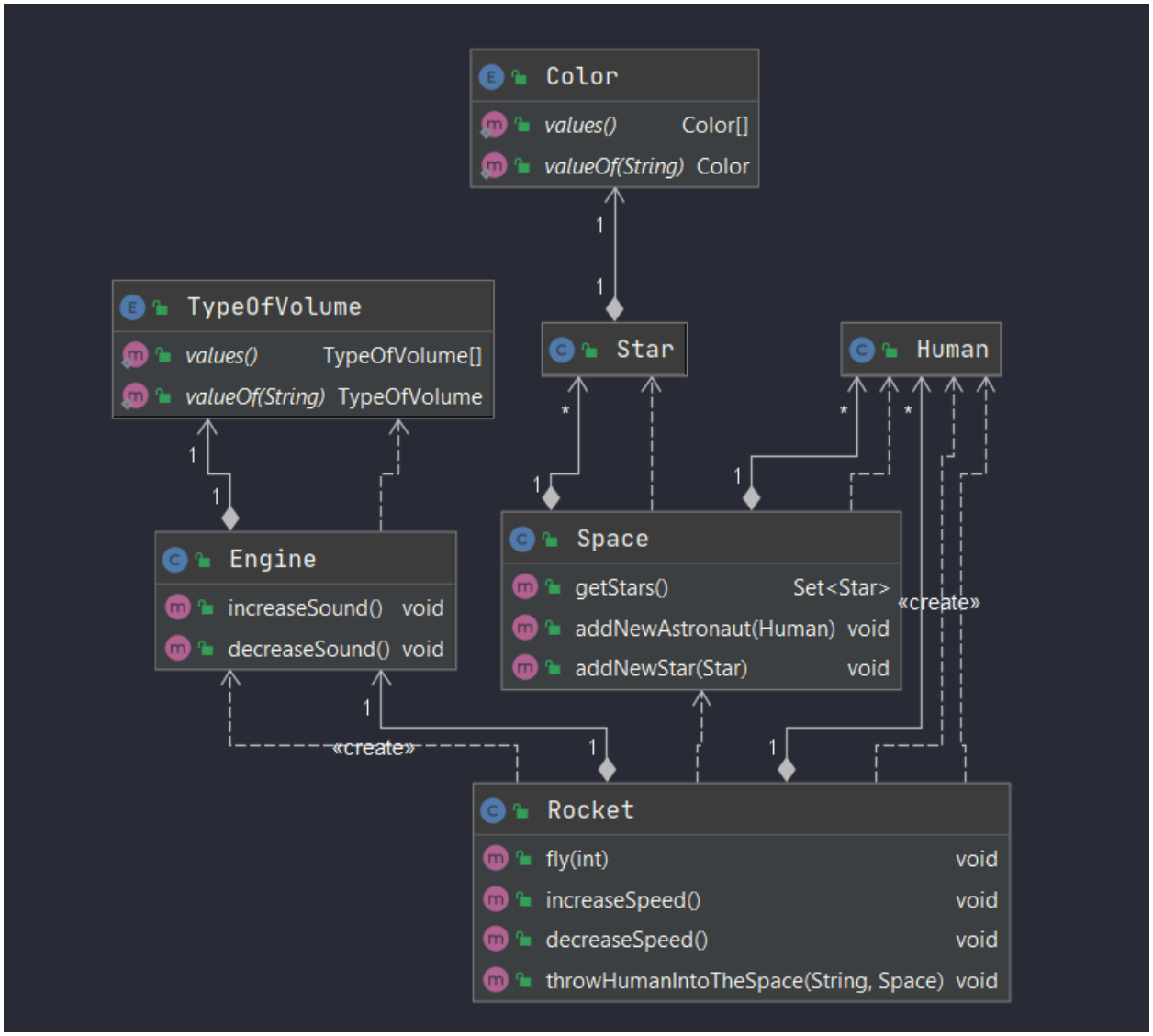
Алгоритм пирамидальной сортировки или HeapSort: проверка происходит по средством собственноручно составленными массивами и через генератор на сайте https://www.cs.usfca.edu/~galles/visualization/HeapSort.html выданный по варианту.
Также нужно не забыть проверить пустой массив, отрицательные числа и повторяющиеся значения.

```java
public class HeapSortTest {
    @Test
    @DisplayName("Check positive values")
    void checkSorting() {
        assertAll(
                () -> assertArrayEquals(new int[]{3, 4, 5, 33}, HeapSort.sort(new int[]{33, 3, 4, 5})),
                () -> assertArrayEquals(new int[]{1,3,4,7,9}, HeapSort.sort(new int[]{4,1,3,9,7})),
                () -> assertArrayEquals(new int[]{1,2,3,4,5,6,7,8,9,10}, HeapSort.sort(new int[]{10,9,8,7,6,5,4,3,2,1})),
                () -> assertArrayEquals(new int[]{22, 28, 30, 82, 159, 166, 166, 287, 289, 338, 365, 371, 444, 451, 520, 527, 567, 623, 676, 714, 719, 741, 771, 783, 818, 874, 884, 914, 960, 970, 983},
                        HeapSort.sort(new int[]{444, 520, 166, 874, 676, 22, 338, 783, 365, 970, 623, 30, 287, 451, 914, 371, 741, 818, 527, 714, 166, 159, 960, 289, 567, 82, 771, 983, 719, 884, 28}))
                );
    }

    @Test
    @DisplayName("Check empty")
    void checkEmpty() {
        assertAll(
                () -> assertArrayEquals(new int[]{}, HeapSort.sort(new int[]{}))
        );
    }

    @Test
    @DisplayName("Check negative values")
    void checkNegativeValues() {
        assertAll(
                () -> assertArrayEquals(new int[]{-5, -4, -3, -2, -1}, HeapSort.sort(new int[]{-5, -1, -4, -3, -2}))
        );
    }

    @Test
    @DisplayName("Check zeros")
    void checkZeros() {
        assertAll(
                () -> assertArrayEquals(new int[]{0,0,0,0,0}, HeapSort.sort(new int[]{0,0,0,0,0}))
        );
    }
}
```

Как мы можем видеть, все работает корректно



Unit-тестирование объектов



Тестирование методы классов

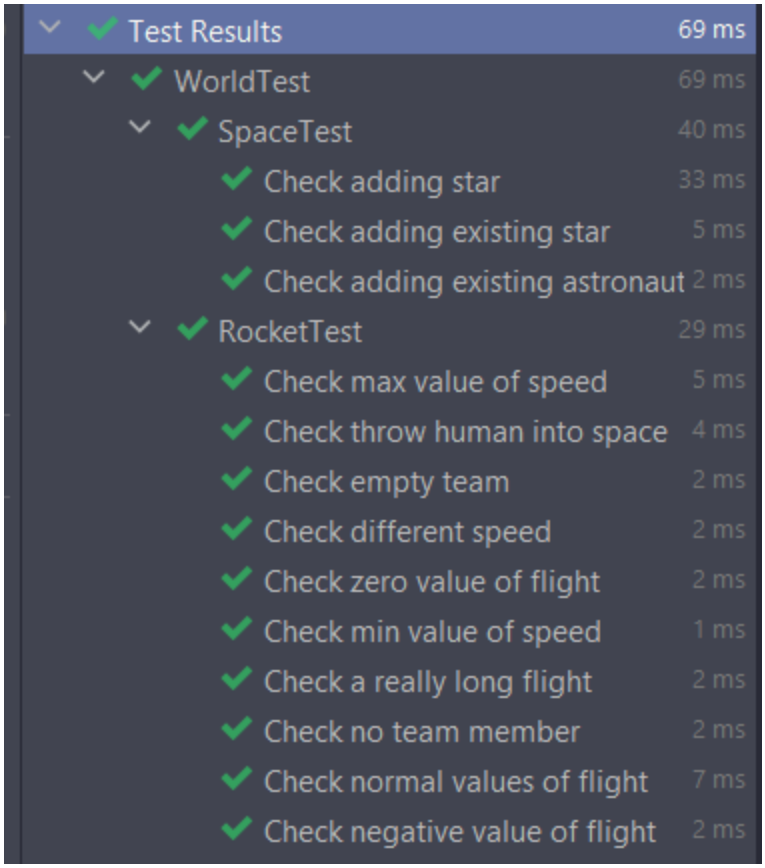
```java
public class WorldTest {

    @Nested
    class RocketTest {

        private Space space;
        private Set<Human> humans;
        private Rocket rocket;

        @BeforeEach
        void init() {
            space = new Space();
            humans = new HashSet<>(Arrays.asList(new Human( name: "Форд"), new Human( name: "Артур")));
            rocket = new Rocket(humans);
        }

        @Test
        @DisplayName("Check a really long flight")
        void checkTooBigValueToFly() {
            Throwable exception = assertThrows(Exception.class, () -> rocket.fly( value: 101));
            assertEquals( expected: "Топлива не хватит на осуществление полета!", exception.getMessage());
        }

        @Test
        @DisplayName("Check negative value of flight")
        void checkNegativeValueOfFlight() {
            Throwable exception = assertThrows(Exception.class, () -> rocket.fly( value: -1));
            assertEquals( expected: "Нельзя пролететь отрицательное расстояние!", exception.getMessage());
        }
```

```java
@Test
@DisplayName("Check zero value of flight")
void checkZeroValueOfFlight() {
    Throwable exception = assertThrows(Exception.class, () -> rocket.fly( value: 0));
    assertEquals( expected: "Нельзя пролететь отрицательное расстояние!", exception.getMessage());
}

@Test
@DisplayName("Check normal values of flight")
void checkNormalFlight() {
    assertAll(
            () -> {
                rocket.fly( value: 7);
                assertEquals( expected: 93, rocket.getFuel());
            },
            () -> {
                rocket.fly( value: 23);
                assertEquals( expected: 70, rocket.getFuel());
            },
            () -> {
                rocket.fly( value: 70);
                assertEquals( expected: 0, rocket.getFuel());
            }
    );

    Throwable exception = assertThrows(Exception.class, () -> rocket.fly( value: 0));
    assertEquals( expected: "Нельзя пролететь отрицательное расстояние!", exception.getMessage());
}

@Test
@DisplayName("Check max value of speed")
void checkMaxSpeed() {
```

```
void checkMaxSpeed() {
    for (int i = 0; i <= 3; i++) rocket.increaseSpeed();
    Throwable exception = assertThrows(Exception.class, () -> rocket.increaseSpeed());
    assertEquals( expected: "Громче двигатель не работает!", exception.getMessage());
}

@Test
@DisplayName("Check min value of speed")
void checkMinSpeed() {
    Throwable exception = assertThrows(Exception.class, () -> rocket.decreaseSpeed());
    assertEquals( expected: "Тише двигатель не работает!", exception.getMessage());
}

@Test
@DisplayName("Check different speed")
void checkDifferentSpeed() {
    rocket.increaseSpeed();
    rocket.increaseSpeed();
    rocket.increaseSpeed();
    rocket.decreaseSpeed();
    rocket.decreaseSpeed();
    assertEquals( expected: 100, rocket.getSpeed());
}

@Test
@DisplayName("Check throw human into space")
void checkThrowingHumanIntoTheSpace() {
    rocket.throwHumanIntoTheSpace( name: "Форд", space);
    assertFalse(rocket.getTeam().contains(new Human( name: "Форд")));
}
```

```
    @Test
    @DisplayName("Check empty team")
    void checkEmptyTeam() {
        humans = new HashSet<>();
        rocket = new Rocket(humans);
        Throwable exception = assertThrows(Exception.class, () -> rocket.throwHumanIntoTheSpace( name: "Елизавета", space));
        assertEquals( expected: "На корабле нет экипажа!", exception.getMessage());
    }

    @Test
    @DisplayName("Check no team member")
    void checkNoTeamMember() {
        Throwable exception = assertThrows(Exception.class, () -> rocket.throwHumanIntoTheSpace( name: "Елизавета", space));
        assertEquals( expected: "Такого человека нет на борту!", exception.getMessage());
    }
}
```

```java
@Nested
class SpaceTest {

    private Set<Human> humans;
    private Space space;

    @BeforeEach
    void init() {
        space = new Space();
        humans = new HashSet<>(Arrays.asList(new Human( name: "Форд"), new Human( name: "Артур")));
    }

    @Test
    @DisplayName("Check adding existing astronaut")
    void checkAddExistAstronaut() {
        space.addNewAstronaut(new Human( name: "Лена"));
        Throwable exception = assertThrows(Exception.class, () -> space.addNewAstronaut(new Human( name: "Лена")));
        assertEquals( expected: "Этот человек уже летает в космосе!", exception.getMessage());
    }

    @Test
    @DisplayName("Check adding star")
    void checkAddStar() {
        space.addNewStar(new Star(Color.BLUE));
        space.addNewStar(new Star(Color.GREEN));
        space.addNewStar(new Star(Color.RED));
        assertEquals( expected: 3, space.getStars().size());
    }

    @Test
    @DisplayName("Check adding existing star")
    void checkAddExistStar() {
        space.addNewStar(new Star(Color.BLUE));
        Throwable exception = assertThrows(Exception.class, () -> space.addNewStar(new Star(Color.BLUE)));
        assertEquals( expected: "Такая звезда уже существует!", exception.getMessage());
    }
}
```



Вывод

В процессе выполнения лабораторной работы мы научился писал юнит-тесты (метод черного ящика) для разработанных классов. Сложность заключается в необходимости проявить гибкость мышления при проверке ожидаемого поведения, т.е. придумать альтернативный способ 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.