Национальный исследовательский университет ИТМО

Факультет ПИиКТ

Лабораторная работа №1

по дисциплине
«Тестирование программного обеспечения»

Вариант №21796

Работу выполнила:

Тройникова В.Д.

Группа: P33302

Преподаватель:

Гаврилов А.В.

Санкт-Петербург

2024

Задание

1. Для указанной функции провести модульное тестирование разложения функции в степенной ряд. Выбрать достаточное тестовое покрытие. Заданная функция - *arcsin(x)*.
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.

   Программный модуль для работы с хеш-таблицей с закрытой адресацией *(Hash Integer, http://www.cs.usfca.edu/~galles/visualization/OpenHash.html)*.

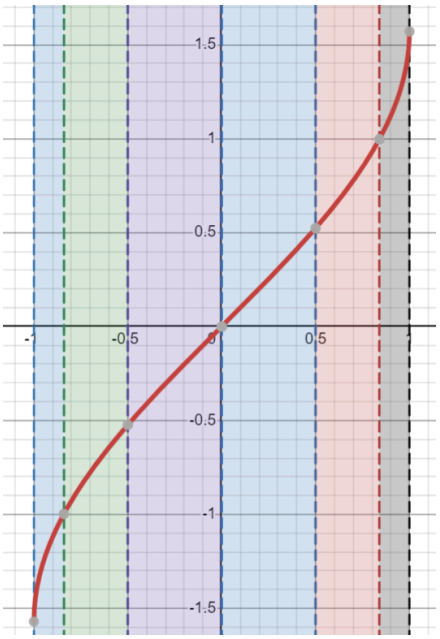
3. Сформировать доменную модель для заданного текста. Разработать тестовое покрытие для данной доменной модели. Описание предметной области:

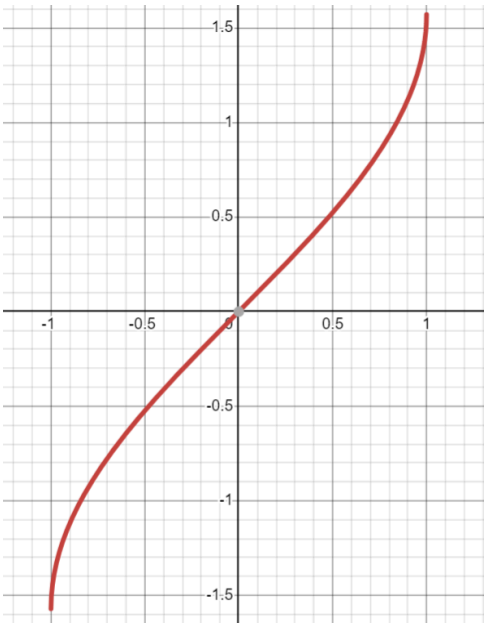
   Поскольку Форд так и не научился произносить свое настоящее имя, его отец, в конце концов, умер от стыда, который в некоторых частях Галактики все еще является смертельной болезнью. В школе Форду дали прозвище Ыкс, что на языке Бетельгейзе-5 означает: "мальчик, который не может внятно объяснить, что такое Хрунг, и почему он решил сотрястись именно на Бетельгейзе-7".

Выполнение

Часть 1

Для тестирования функции arcsin(x) проведем анализ эквивалентности:





Всего было выделено 6 областей, значения для тестирования будут представлены в виде граничных значений и значений из данных областей:

Граничные значения	*Значения из областей*
-1	-0.93
-0.84	-0.65
-0.5	-0.34
0	0.28
0.5	0.67
0.84	0.9
1	

Разбиение на области основано на изменении знака функции и скорости роста функции.

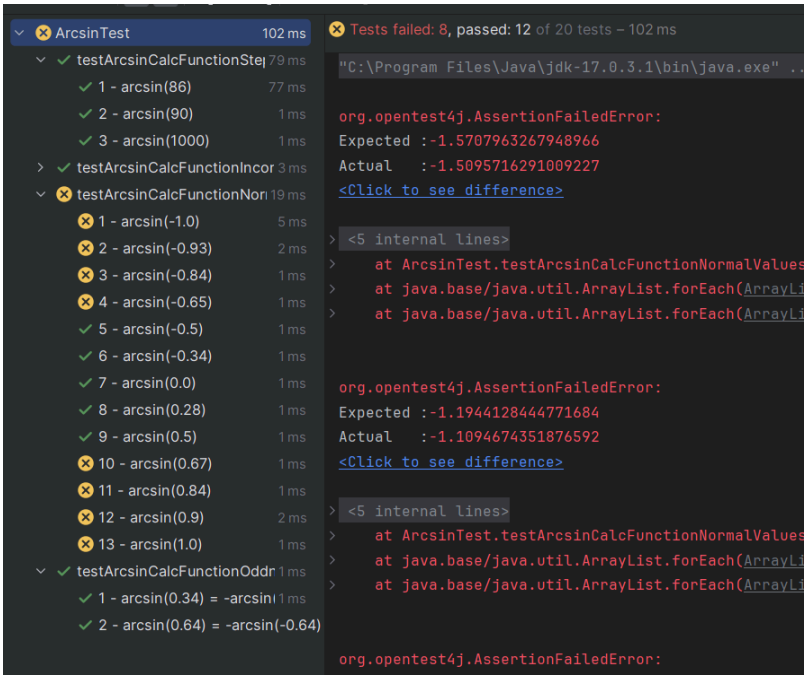
Разложение функции arcsin(x) в ряд Тейлора:

$$\arcsin x = x + \frac{x^3}{6} + \frac{3x^5}{40} + \cdots = \sum_{n=0}^{\infty} \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1} \qquad |x| < 1$$

Код тестов:

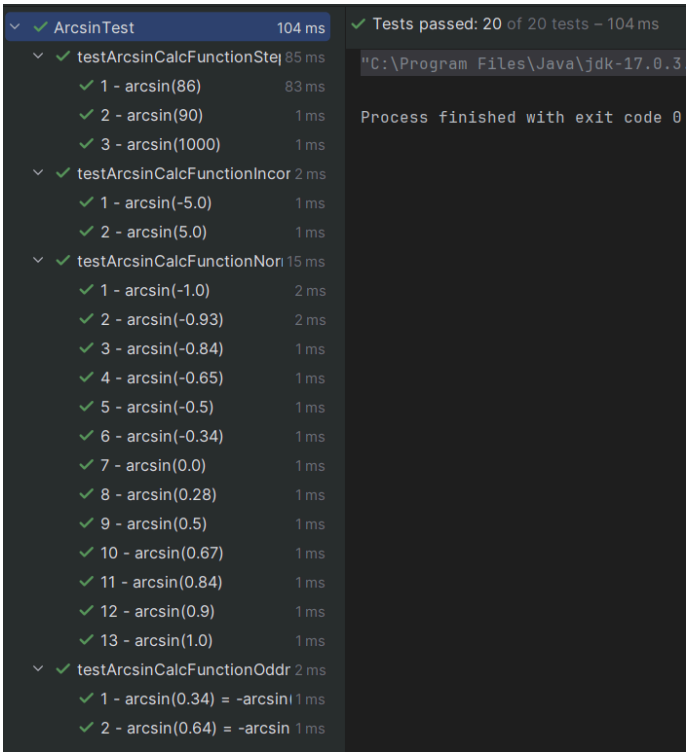
```java
public class ArcsinTest {
    @ParameterizedTest(name = "{index} - arcsin({0})")
    @ValueSource(doubles = {-1, -0.93, -0.84, -0.65, -0.5, -0.34, 0, 0.28, 0.5, 0.67, 0.84, 0.9, 1})
    public void testArcsinCalcFunctionNormalValues(double number) {
        assertEquals(Math.asin(number), calc_arcsin(number, count: 85), delta: 0.01);
    }

    @ParameterizedTest(name = "{index} - arcsin({0})")
    @ValueSource(doubles = {-5, 5})
    public void testArcsinCalcFunctionIncorrectValues(double number) {
        assertEquals(Double.NaN, calc_arcsin(number, count: 80), delta: 0.1);
    }

    @ParameterizedTest(name = "{index} - arcsin({0})")
    @ValueSource(ints = {86, 90, 1000})
    public void testArcsinCalcFunctionStepMoreThan85(int count) {
        assertEquals(Double.POSITIVE_INFINITY, calc_arcsin( x: 0.5, count), delta: 0.1);
    }

    @ParameterizedTest(name = "{index} - arcsin({0}) = -arcsin(-{0})")
    @ValueSource(doubles = {0.34, 0.64})
    public void testArcsinCalcFunctionOddness(double number) {
        assertEquals( -calc_arcsin(number, count: 80), calc_arcsin(-number, count: 80), delta: 0.0001);
    }
}
```

Результаты выполнения тестов:



Как видно из результатов тестирования, области 1,2,4 и 5 не проходят тестирования с погрешностью 0.01. Это связано с тем, что разложение в степенной ряд плохо представляет arcsin при его крайних значениях.

При значении точности 0.1 проходят все тесты:



Часть 2

Для хеш-таблицы были реализованы операции вставки, поиска и удаления. Основываясь на алгоритмах данных действий были выявлены следующие характерные точки:

Вставка	*Поиск*	*Удаление*
Вставка элемента в пустую цепочку	Поиск несуществующего элемента	Удаление несуществующего элемента
Вставка элемента в начало существующей цепочки	Поиск элемента в начале цепочки	Удаление элемента из начала цепочки
	Поиск элемента в середине цепочки	Удаление элемента из середины цепочки
	Поиск элемента в конце цепочки	Удаление элемента из конца цепочки
		Удаление единственного элемента в цепочке

Таблица получившихся тестовых случаев:

Предусловия (состояние таблицы)	*Входные значения*	*Ожидаемый результат*
[[10][null][null][null][14,9,4]]	поиск числа 5	false; [[10][null][null][null][14,9,4]]
[[10][null][null][null][14,9,4]]	поиск числа 14	true; [[10][null][null][null][14,9,4]]
[[10][null][null][null][14,9,4]]	поиск числа 9	true; [[10][null][null][null][14,9,4]
[[10][null][null][null][14,9,4]]	поиск числа 4	true; [[10][null][null][null][14,9,4]]
[[10][null][null][null][14,9,4]]	удаление числа 5	false; [[10][null][null][null][14,9,4]]
[[10][null][null][null][14,9,4]]	удаление числа 14	true; [[10][null][null][null][9,4]]
[[10][null][null][null][14,9,4]]	удаление числа 9	true; [[10][null][null][null][14,4]]
[[10][null][null][null][14,9,4]]	удаление числа 4	true; [[10][null][null][null][14,9]]
[[10][null][null][null][14,9,4]]	удаление числа 10	true; [[null][null][null][null][14,9,4]]
[[10][null][null][null][14,9,4]]	вставка числа 4	[[10][null][null][null][4,14,9,4]]

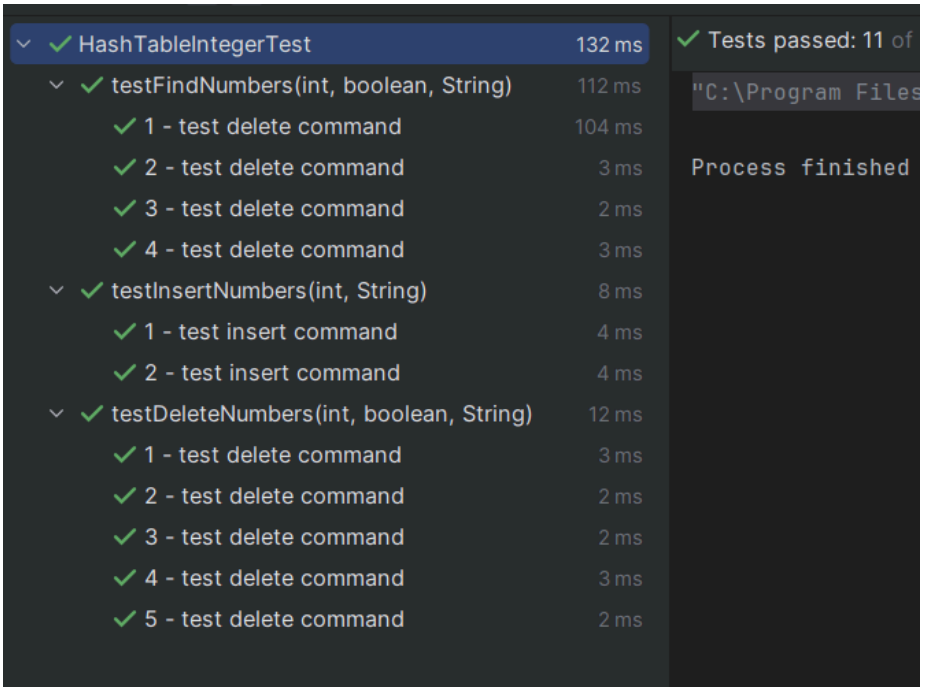
[10, null, null, null, 14, 9, 4]	вставка числа 1	[10, 1, null, null, 14, 9, 4]

Код тестов:

```
Troynikova Veronika *
public class HashTableIntegerTest {

    11 usages
    private HashTableInteger table;

    Troynikova Veronika
    @BeforeEach
    void init() {
        table = new HashTableInteger( init: 5);
        table.add_element( value: 4);
        table.add_element( value: 9);
        table.add_element( value: 14);
        table.add_element( value: 10);
    }

    Troynikova Veronika *
    @ParameterizedTest(name = "{index} - test insert command")
    @CsvSource({
            "4, '[[10][null][null][null][4,14,9,4]]'",
            "1, '[[10][1][null][null][14,9,4]]'"
    })
    public void testInsertNumbers(int number, String expectedArray) {
        table.add_element(number);
        assertEquals(expectedArray, table.getAsString());
    }
```

```
    Troynikova Veronika *
    @ParameterizedTest(name = "{index} - test delete command")
    @CsvFileSource(resources = "/delete_values.csv", numLinesToSkip = 1, delimiter = ';')
    public void testDeleteNumbers(int number, boolean result, String expectedArray) {
        boolean cur_result = table.delete_element(number);
        assertEquals(result, cur_result);
        assertEquals(expectedArray, table.getAsString());
    }

    Troynikova Veronika *
    @ParameterizedTest(name = "{index} - test find command")
    @CsvFileSource(resources = "/find_values.csv", numLinesToSkip = 1, delimiter = ';')
    public void testFindNumbers(int number, boolean result, String expectedArray) {
        boolean cur_result = table.find_element(number);
        assertEquals(result, cur_result);
        assertEquals(expectedArray, table.getAsString());
    }
}
```

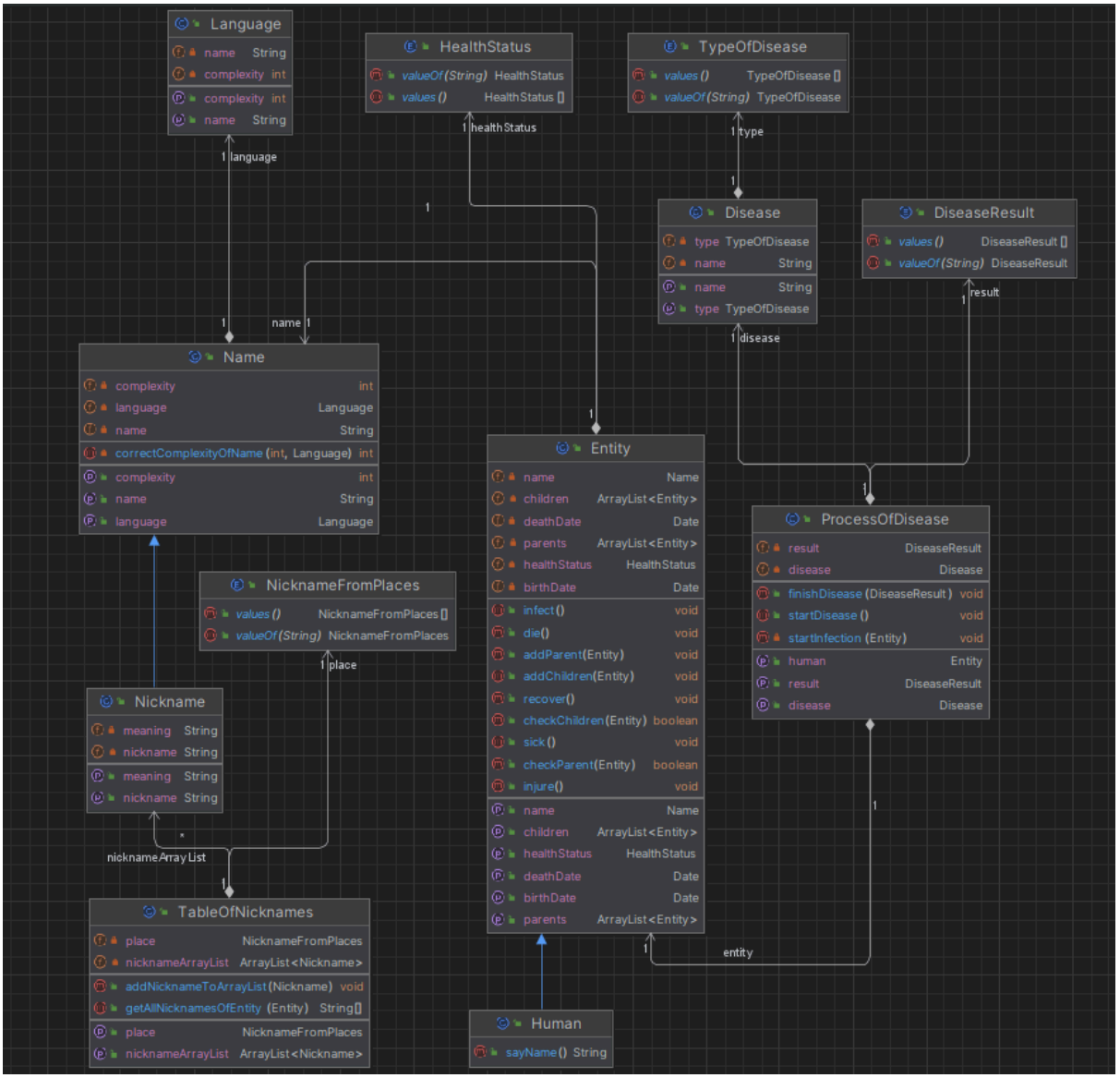
Результаты выполнения тестов:



Анализируя результаты тестирования можно сделать вывод о том, что алгоритм работает корректно.

Часть 3

На основе заданного теста была сформирована следующая доменная модель:



Для тестирования были отобраны классы, содержащие сложную логику (пропущены классы, содержащие только get/set методы). Тестовое покрытие было разработано исходя из ограничений и логических переходов, предусмотренных моделью.

Код тестов:

```java
public class ModelTest {
    @Nested
    class ProcessOfDiseaseClassTest {
        10 usages
        private Entity entity;
        2 usages
        private Disease simpleDisease;
        2 usages
        private Disease deadlyDisease;
        @BeforeEach
        void init() {
            Language l1 = null;
            try {
                l1 = new Language( name: "Бетельгейзе-5",  complexity: 3);
            } catch (Exception ignored) {}
            entity = new Entity(new Name( name: "Форд",l1), new Date());
            simpleDisease = new Disease( name: "Простуда", TypeOfDisease.PASSING_WITHOUT_TREATMENT);
            deadlyDisease = new Disease( name: "Стыд", TypeOfDisease.DEADLY);
        }

        @Test
        @DisplayName("Check health type of entity without disease")
        void testHealthTypeOfEntityWithoutDisease() {
            assertEquals(HealthStatus.HEALTHFUL, entity.getHealthStatus());
        }
```

```java
@ParameterizedTest
@ArgumentsSource(ProcessOfDiseaseProvider.class)
@DisplayName("Check health type of entity connect to disease")
void testHealthTypeOfEntityWithDisease(Disease disease, DiseaseResult result, HealthStatus status) {
    ProcessOfDisease processOfDisease = new ProcessOfDisease(disease, entity);
    assertAll(
            () -> {
                assertEquals(HealthStatus.INFECTED, entity.getHealthStatus());
            },
            () -> {
                processOfDisease.startDisease();
                assertEquals(HealthStatus.SICK, entity.getHealthStatus());
            },
            () -> {
                processOfDisease.finishDisease(result);
                assertEquals(status, entity.getHealthStatus());
            }
    );
}
```

```java
@Test
@DisplayName("Check not being able to recover after deadly disease")
void testRecoverAfterDeadlyDisease() {
    ProcessOfDisease processOfDeadlyDisease = new ProcessOfDisease(deadlyDisease, entity);
    assertAll(
            () -> {
                processOfDeadlyDisease.startDisease();
                assertEquals(HealthStatus.SICK, entity.getHealthStatus());
            },
            () -> {

                Throwable exception = assertThrows(Exception.class, () ->  processOfDeadlyDisease.finishDisease(DiseaseResult.WITHOUT_TRACE));
                assertEquals( expected: "Чуда не бывает, нельзя выжить после смертельной болезни!", exception.getMessage());
            },
            () -> {
                Throwable exception = assertThrows(Exception.class, () ->  processOfDeadlyDisease.finishDisease(DiseaseResult.HEALTH_CONSEQUENCES));
                assertEquals( expected: "Чуда не бывает, нельзя выжить после смертельной болезни!", exception.getMessage());
            }
    );
}
```

```java
@Test
@DisplayName("Check not being able to die after simple disease")
void testDieAfterSimpleDisease() {
    ProcessOfDisease processOfSimpleDisease = new ProcessOfDisease(simpleDisease, entity);
    assertAll(
            () -> {
                processOfSimpleDisease.startDisease();
                assertEquals(HealthStatus.SICK, entity.getHealthStatus());
            },
            () -> {

                Throwable exception = assertThrows(Exception.class, () ->  processOfSimpleDisease.finishDisease(DiseaseResult.DEATH));
                assertEquals( expected: "Болезнь не вызывает смерть", exception.getMessage());
            }
    );
}
```

```java
1 usage
static class ProcessOfDiseaseProvider implements ArgumentsProvider {

    no usages
    @Override
    public Stream<? extends Arguments> provideArguments(ExtensionContext extensionContext) {
        return Stream.of(
                Arguments.of(new Disease( name: "Простуда", TypeOfDisease.PASSING_WITHOUT_TREATMENT), DiseaseResult.WITHOUT_TRACE, HealthStatus.HEALTHFUL),
                Arguments.of(new Disease( name: "Простуда", TypeOfDisease.PASSING_WITHOUT_TREATMENT), DiseaseResult.HEALTH_CONSEQUENCES, HealthStatus.INJURY),
                Arguments.of(new Disease( name: "Корь", TypeOfDisease.CURABLE), DiseaseResult.HEALTH_CONSEQUENCES, HealthStatus.INJURY),
                Arguments.of(new Disease( name: "Корь", TypeOfDisease.CURABLE), DiseaseResult.WITHOUT_TRACE, HealthStatus.HEALTHFUL),
                Arguments.of(new Disease( name: "Корь", TypeOfDisease.CURABLE), DiseaseResult.DEATH, HealthStatus.DEAD),
                Arguments.of(new Disease( name: "Стыд", TypeOfDisease.DEADLY), DiseaseResult.DEATH, HealthStatus.DEAD)
        );
    }
}
```

```java
@Nested
class TableOfNicknamesClassTest {
    10 usages
    private TableOfNicknames table;
    4 usages
    private Entity entity;
    8 usages
    private Language l1;
    @BeforeEach
    void init() {
        try {
            l1 = new Language( name: "Бетельгейзе-5",  complexity: 3);
        } catch (Exception ignored) {}
        table = new TableOfNicknames(NicknameFromPlaces.SCHOOL);
        entity = new Entity(new Name( name: "Форд",l1), new Date());
    }
    @Test
    @DisplayName("Check taking names from table of nicknames")
    void testTakingNamesFromTableOfNicknames() {
        table.addNicknameToArrayList(new Nickname( name: "abc", l1,  nickname: "abcd"));
        table.addNicknameToArrayList(new Nickname( name: "Форд", l1,  nickname: "Ыкс"));
        table.addNicknameToArrayList(new Nickname( name: "bcd", l1,  nickname: "bcdf"));
        assertArrayEquals(new String[]{"Ыкс"}, table.getAllNicknamesOfEntity(entity));
    }
```

```java
    @Test
    @DisplayName("Check taking names from empty table of nicknames")
    void testAddingNameWithIncorrectComplexity() {
        assertArrayEquals(new String[]{}, table.getAllNicknamesOfEntity(entity));
    }

    @Test
    @DisplayName("Check taking no names from table of nicknames")
    void testAddingLanguageWithIncorrectComplexity() {
        table.addNicknameToArrayList(new Nickname( name: "abc", l1,  nickname: "abcd"));
        table.addNicknameToArrayList(new Nickname( name: "ggg", l1,  nickname: "Ыкс"));
        table.addNicknameToArrayList(new Nickname( name: "bcd", l1,  nickname: "bcdf"));
        assertArrayEquals(new String[]{}, table.getAllNicknamesOfEntity(entity));
    }
}
```

```java
@Nested
class HumanClassTest {
    5 usages
    private Human h1;
    @BeforeEach
    void init() {
        Language l1 = null;
        try {
            l1 = new Language( name: "Бетельгейзе-7",  complexity: 6);
        } catch (Exception ignored) {}
        h1 = new Human(new Name( name: "Форд", l1), new Date());
    }
    @Test
    @DisplayName("Check say name with different complexity")
    void testSayNameFunc() {
        assertAll(
                () -> {
                    h1.getName().setComplexity(3);
                    assertEquals( expected: "Форд", h1.sayName());
                },
                () -> {
                    h1.getName().setComplexity(6);
                    Throwable exception = assertThrows(Exception.class, () ->  h1.sayName());
                    assertEquals( expected: "Имя не может быть произнесено, оно слишком сложное!", exception.getMessage());
                }
        );
    }
}
```

```java
@Nested
class NameAndLanguageClassTest {
    3 usages
    private Language l1;
    4 usages
    private Language l2;
    @BeforeEach
    void init() {
        try {
            l1 = new Language( name: "Бетельгейзе-5",  complexity: 3);
            l2 = new Language( name: "Бетельгейзе-7",  complexity: 6);
        } catch (Exception ignored) {}
    }
```

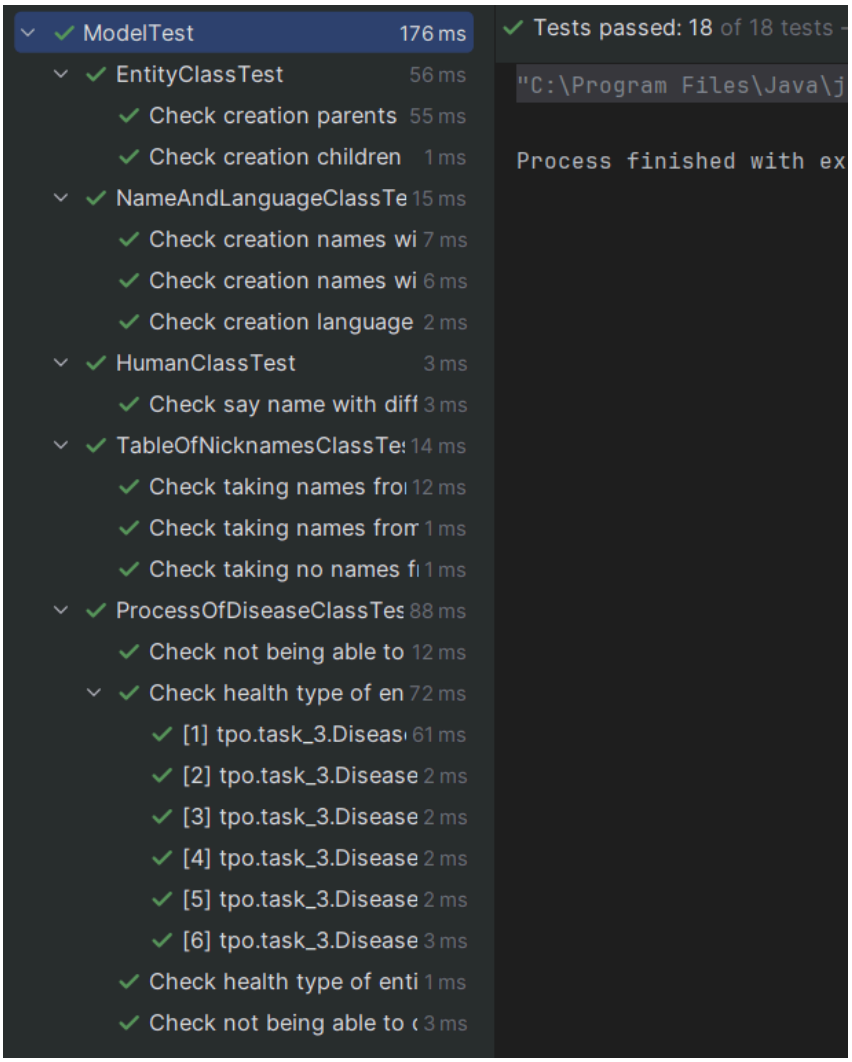
```java
@Test
@DisplayName("Check creation names with normal complexity")
void testAddingNameWithNormalComplexity() {
    assertAll(
            () -> {
                Name name1 = new Name( name: "Форд", l1);
                assertEquals( expected: 5, name1.getComplexity());
            },
            () -> {
                Name name1 = new Name( name: "Форд", l1,  complexity: 6);
                assertEquals( expected: 6, name1.getComplexity());
            },
            () -> {
                Name name1 = new Name( name: "Форд", l2,  complexity: 6);
                assertEquals( expected: 9, name1.getComplexity());
            },
            () -> {
                Name name1 = new Name( name: "Форд", l2,  complexity: 9);
                assertEquals( expected: 10, name1.getComplexity());
            }
    );
}
```

```
    @Test
    @DisplayName("Check creation names with incorrect complexity")
    void testAddingNameWithIncorrectComplexity() {
        Throwable exception = assertThrows(Exception.class, () ->  new Name( name: "Форд", l2,  complexity: 12));
        assertEquals( expected: "Сложность имени должна быть в пределах от 0 до 10!", exception.getMessage());
    }

    @Test
    @DisplayName("Check creation language with incorrect complexity")
    void testAddingLanguageWithIncorrectComplexity() {
        Throwable exception = assertThrows(Exception.class, () ->  new Language( name: "fffff",  complexity: 12));
        assertEquals( expected: "Сложность языка должна быть в пределах от 0 до 10!", exception.getMessage());
    }
}
```

```
@Nested
class EntityClassTest {
    11 usages
    private Entity h1;
    11 usages
    private Entity h2;
    @BeforeEach
    void init() {
        Language l1 = null;
        try {
            l1 = new Language( name: "Бетельгейзе-7",  complexity: 6);
        } catch (Exception ignored) {}
        h1 = new Entity(new Name( name: "Форд", l1), new Date());
        h2 = new Entity(new Name( name: "Джон", l1), new Date());
    }
    @Test
    @DisplayName("Check creation parents")
    void testAddingParentsToHuman() {
        h1.addParent(h2);
        assertTrue(h1.checkParent(h2));
        assertTrue(h2.checkChildren(h1));
        assertFalse(h2.checkParent(h1));
        assertFalse(h1.checkChildren(h2));
    }
```

```
        @Test
        @DisplayName("Check creation children")
        void testAddingChildrenToHuman() {
            h1.addChildren(h2);
            assertTrue(h2.checkParent(h1));
            assertTrue(h1.checkChildren(h2));
            assertFalse(h1.checkParent(h2));
            assertFalse(h2.checkChildren(h1));
        }
    }
}
```

Результаты выполнения тестов:



Все тесты пройдены успешно

Вывод:

В ходе выполнения данной лабораторной работы были изучены такие темы, как использование тестирование ПО и модульное тестирование ПО. В процессе выполнения задания был использован JUnit5 и его основные аннотации.