

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО”

Дисциплина

Тестирование программного обеспечения

Лабораторная работа №1

Вариант: 368273

Выполнили:
Карандашева Анастасия Денисовна
Ястребов-Амирханов Алекси
Группа: Р3332

Преподаватель:
Чупанов Аликылыч Алибекович

г. Санкт-Петербург

2026

Содержание

Задание	3
Часть 1	3
Точки для тестирования.....	4
Таблица ожидаемых значений	4
Тесты разложения $\cos(x)$	5
Результаты тестирования.....	6
Часть 2	7
Планируемые к покрытию операции и свойства	8
1. Поиск минимального элемента (min / getMinimum).....	8
2. Добавление элемента (insert)	8
3. Удаление минимального элемента (removeMin).....	8
4. Объединение куч (merge)	9
5. Уменьшение ключа (decreaseKey).....	9
Структура тестов	9
Тесты для фибоначчиевой кучи	10
Результаты тестирования.....	15
Часть 3	16
Описание классов и их методов.....	16
Структура тестов	18
Тесты доменной модели	19
Результаты тестирования.....	22
Выводы.....	23

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 368273:

1. Функция $\cos(x)$
2. Программный модуль для работы с Фибоначчиевой кучей (Internal Representation, <http://www.cs.usfca.edu/~galles/visualization/FibonacciHeap.html>)
3. Описание предметной области:
После недолгой поездки на аэромобиле Артур и старый магратеянин оказались у какой-то двери. Они вышли из машины и прошли в приемную, уставленную стеклянными столиками и наградами из прозрачного пластика. Почти сразу же над дверью в другом конце комнаты загорелся свет, и они вошли в нее.

Часть 1

Реализация разложения функции $\cos(x)$:

[ITMO/ТПО/Лаб1/Code/TPOLAB1/src/main/java/org/example/math/Cosine.java](https://github.com/ITMO/ТПО/Лаб1/Code/TPOLAB1/src/main/java/org/example/math/Cosine.java) at master · AlexandroLe/ITMO

В коде используется разложение функции косинуса в ряд Тейлора (ряд Маклорена) в точке 0:

$$\cos x = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!}$$

Первые члены ряда:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \dots$$

В реализации используется рекуррентная формула для вычисления следующего члена:

$$a_k = a_{k-1} \cdot \frac{-x^2}{(2k-1)(2k)}$$

Точки для тестирования

- **0.0** – базовая точка разложения (центр ряда Маклорена); значение должно вычисляться ровно как 1 без накопления погрешности и без лишних итераций;
- **0.000001, -0.000001** – близкие к 0 значения; должны корректно вычисляться как числа, близкие к 1, а не округляться к 1 преждевременно; проверяется точность работы с малыми аргументами и сохранение чётности функции;
- **1×10^{-308}** – экстремально малое положительное значение; должно корректно обрабатываться без превращения в 0 и без потери точности при возведении в квадрат;
- **$\pi/2, -\pi/2$** – граничные точки четверти периода; функция должна возвращать значение, близкое к 0, корректно проходя через ветвление редукции аргумента;
- **$\pi, -\pi$** – граничные точки полу периода; должны корректно редуцироваться по модулю 2π и возвращать значение -1 с достаточной точностью;
- **$3\pi/4$** – значение во второй четверти; проверяется корректность отражения аргумента в диапазон $[-\pi/2, \pi/2]$ и правильность смены знака результата;
- **1000000, -1000000** – большие по модулю значения; проверяется корректность операции взятия остатка по 2π и устойчивость редукции аргумента при больших входных данных;
- **2π** – значение, кратное периоду; должно определяться как 1 с высокой точностью, подтверждая корректность периодичности функции;
- **Double.NaN** – должно возвращать NaN, как в эталонной реализации Math.cos;
- **Double.POSITIVE_INFINITY, Double.NEGATIVE_INFINITY** – значения вне допустимой области вычисления; должны возвращать NaN, аналогично эталонной реализации.

Таблица ожидаемых значений

x	cos(x)
0.0	1.0
0.000001	0.9999999999995
-0.000001	0.9999999999995
1×10^{-308}	1.0
$\pi/2$	0.0
$-\pi/2$	0.0
π	-1.0
$-\pi$	-1.0
$3\pi/4$	-0.7071067811865476
1000000	0.9367521275331447
-1000000	0.9367521275331447
2π	1.0
NaN	NaN
+Infinity	NaN
-Infinity	NaN

Тесты разложения $\cos(x)$

```
package org.example.math;

import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.MethodSource;
import org.junit.jupiter.params.provider.Arguments;

import java.util.stream.Stream;

import static org.junit.jupiter.api.Assertions.*;

class CosineParameterizedTest {

    @ParameterizedTest(name = "[{index}] x={0}, expect={1}, tol={2}")
    @MethodSource("valuesProvider")
    void parameterizedCosTests(double x, Double expected, double tol) {
        double actual = Cosine.cos(x);

        if (expected == null) {
            assertTrue(Double.isNaN(actual), () -> "Expected NaN for input: " + x +
", but was: " + actual);
        } else {
            assertEquals(expected, actual, tol, () -> String.format("x=%.18g,
expected=%.18g, actual=%.18g", x, expected, actual));
        }
    }

    static Stream<Arguments> valuesProvider() {
        return Stream.of(
            Arguments.of(0.0, Math.cos(0.0), 1e-15),

            Arguments.of(1e-6, Math.cos(1e-6), 1e-12),
            Arguments.of(-1e-6, Math.cos(-1e-6), 1e-12),

            Arguments.of(1e-308, Math.cos(1e-308), 1e-15),

            Arguments.of(Math.PI / 2.0, Math.cos(Math.PI / 2.0), 1e-12),
            Arguments.of(-Math.PI / 2.0, Math.cos(-Math.PI / 2.0), 1e-12),
            Arguments.of(Math.PI, Math.cos(Math.PI), 1e-12),
            Arguments.of(-Math.PI, Math.cos(-Math.PI), 1e-12),
            Arguments.of(3.0 * Math.PI / 4.0, Math.cos(3.0 * Math.PI / 4.0),
1e-12),

            Arguments.of(1e6, Math.cos(1e6), 1e-9),
            Arguments.of(-1e6, Math.cos(-1e6), 1e-9),

            Arguments.of(2.0 * Math.PI, Math.cos(2.0 * Math.PI), 1e-15),

            Arguments.of(Double.NaN, null, 0.0),
```

```

        Arguments.of(Double.POSITIVE_INFINITY, null, 0.0),
        Arguments.of(Double.NEGATIVE_INFINITY, null, 0.0)
    );
}
}

```

Результаты тестирования

```

-----
T E S T S
-----
Running org.example.math.CosineParameterizedTest
Tests run: 15, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.089 s - in org.example.math.CosineParameterizedTest

Results:

Tests run: 15, Failures: 0, Errors: 0, Skipped: 0

```

Функция $\cos(x)$ корректно обрабатывает все протестированные случаи, включая граничные точки периода, большие по модулю значения и специальные значения типа NaN и бесконечностей. Все результаты находятся в пределах заданной точности и совпадают с эталонной реализацией `Math.cos`.

Реализация правильно выполняет редукцию аргумента по модулю (2π), корректно использует симметрию функции для приведения значения к интервалу $([-\pi/2, \pi/2])$, а также корректно изменяет знак результата при отражении. Это подтверждается тестами в точках $(\pi/2)$, (π) , $(3\pi/4)$, (2π) , а также для больших значений (± 1000000).

Функция корректно возвращает NaN для `Double.NaN`, `Double.POSITIVE_INFINITY` и `Double.NEGATIVE_INFINITY`, что полностью соответствует поведению `Math.cos`.

Проверка малых значений (включая очень малые числа, близкие к пределам точности `double`) показала устойчивость степенного разложения: быстрая сходимость ряда не приводит к накоплению заметной ошибки.

В целом тестирование подтверждает численную устойчивость алгоритма, корректность редукции аргумента и соответствие результатов эталонной реализации в рамках заданных допусков.

Часть 2

Реализация по варианту:

[ITMO/ТПО/Лаб1/Code/TPOLAB1/src/main/java/org/example/heap/Fibonacci.java at master · AlexandroLe/ITMO](#)

Фибоначчиева куча (англ. Fibonacci heap) – структура данных, отвечающая интерфейсу priority queue. Имеет меньшую амортизированную сложность, чем такие приоритетные очереди как биномиальная куча и двоичная куча.

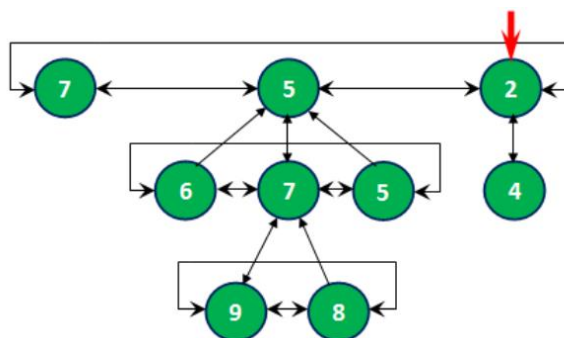


Рисунок 1. Схема фибоначчиевой кучи

Фибоначчиева куча представляет собой набор фибоначчиевых деревьев.

Фибоначчиево дерево – это k -ичное дерево, для каждого элемента которого выполняется правило: «дочерний элемент не превышает своего родителя». Корни фибоначчиевых деревьев хранятся в виде кольцевого списка. Каждый узел фибоначчиева дерева, помимо указателей на левого, правого брата, на родителя и на одного из своих сыновей содержит информацию о количестве дочерних узлов.

Основные операции:

- Поиск минимального элемента (*getMinimum*)
- Добавление нового элемента (*insert*)
- Объединение двух фибоначчиевых куч (*merge*)
- Удаление минимального элемента (*removeMinimum*)
- Изменение (приоритета) элемента (*decreaseKey*)

Целью модульного тестирования будет покрытие характерных точек внутри алгоритма и проверка в них корректности свойств структуры данных.

Планируемые к покрытию операции и свойства

1. Поиск минимального элемента (min / getMinimum)

Планируется проверить:

- Корректность возврата минимального элемента
- Корректность работы на:
 - пустой куче
 - куче из одного элемента
 - куче после merge
 - куче после decreaseKey
 - куче после серии removeMin
- Корректность обновления минимума после каждой модифицирующей операции

2. Добавление элемента (insert)

Планируется проверить:

- Вставку в пустую кучу
- Вставку нескольких элементов подряд
- Вставку:
 - в возрастающем порядке
 - в убывающем порядке
 - в случайном порядке
 - с дубликатами
- Корректность обновления указателя на минимум
- Работоспособность при больших объёмах данных
- Поведение после полной очистки кучи

3. Удаление минимального элемента (removeMin)

Планируется проверить:

- Удаление единственного элемента
- Удаление из пустой кучи
- Последовательное удаление всех элементов
- Сохранение упорядоченности извлечения

- Корректность обновления минимума
- Работу при большом количестве элементов
- Поведение после merge
- Возможность повторного использования кучи после полного опустошения

4. Объединение куч (merge)

Планируется проверить:

- Объединение двух непустых куч
- Объединение непустой и пустой кучи
- Объединение пустой и непустой
- Корректность обновления минимума
- Сохранение корректного порядка извлечения после объединения

5. Уменьшение ключа (decreaseKey)

Планируется проверить:

- Уменьшение ключа с образованием нового минимума
- Уменьшение ключа без структурного разреза
- Уменьшение ключа с каскадным разрезом
- Корректность обновления указателя на минимум

Структура тестов

1. testSequentialInsert5

Проверяет корректность последовательной вставки элементов в возрастающем порядке и правильное определение минимального элемента после вставок.

2. testInsert100000

Проверяет корректность работы кучи при большом объёме данных (100000 элементов), правильность последовательного извлечения минимума и устойчивость структуры при массовых операциях.

3. testReversedInsert

Проверяет корректность работы кучи при вставке элементов в убывающем порядке и сохранение свойства минимальной кучи при последующем извлечении.

4. testShuffledInsert

Проверяет корректность работы кучи при вставке элементов в случайном порядке и правильность восстановления отсортированной последовательности через removeMin.

5. testDuplicates

Проверяет корректность обработки дубликатов, сохранение их количества и правильный порядок извлечения одинаковых значений.

6. testMerge

Проверяет корректность объединения двух непустых куч, правильное определение нового минимума и корректность дальнейшего извлечения элементов из объединённой структуры.

7. testDecreaseKeyNewMin

Проверяет корректность операции decreaseKey при уменьшении значения до нового глобального минимума.

8. testDecreaseKeyCascade

Проверяет корректность работы decreaseKey в ситуации, потенциально вызывающей каскадное отсечение узлов, и правильное обновление минимума.

9. testStructureAfterOperations

Проверяет корректность поддержания структуры кучи при смешанных операциях (insert, min, removeMin) и правильное обновление минимального элемента.

10. testMinOnEmptyHeap

Проверяет корректное поведение метода min() на пустой куче (возврат null).

11. testRemoveMinOnEmptyHeap

Проверяет корректное поведение removeMin() на пустой куче (отсутствие исключений и сохранение пустого состояния).

12. testSingleElementInsertRemove

Проверяет корректность работы кучи при наличии одного элемента и правильное опустошение структуры после его удаления.

13. testMergeWithEmptyHeap

Проверяет корректность объединения непустой кучи с пустой и пустой с непустой, а также правильность определения минимума после объединения.

14. testDecreaseKeyWithoutCut

Проверяет корректность decreaseKey в случае, когда уменьшение ключа не приводит к структурным изменениям (без каскадного отсечения).

15. testReuseAfterFullClear

Проверяет возможность повторного использования кучи после полного удаления всех элементов и корректность работы при новой серии вставок.

Тесты для фибоначчиевой кучи

```
package org.example.heap;

import org.junit.jupiter.api.Test;
import java.time.Duration;
import java.util.*;

import static org.junit.jupiter.api.Assertions.*;
```

```

import static org.junit.jupiter.api.Assertions.assertTimeoutPreemptively;

public class FibonacciTest {

    @Test
    void testSequentialInsert5() {
        assertTimeoutPreemptively(Duration.ofSeconds(5), () -> {
            FibonacciHeap heap = new FibonacciHeap();
            for (int i = 10; i <= 15; i++) {
                heap.insert(i);
            }
            assertNotNull(heap.min());
            assertEquals(10, heap.min().key);
        });
    }

    @Test
    void testInsert100000() {
        assertTimeoutPreemptively(Duration.ofSeconds(5), () -> {
            FibonacciHeap heap = new FibonacciHeap();
            for (int i = 1; i <= 100000; i++) {
                heap.insert(i);
            }
            for (int i = 1; i <= 100000; i++) {
                assertEquals(i, heap.min().key);
                heap.removeMin();
            }
        });
    }

    @Test
    void testReversedInsert() {
        assertTimeoutPreemptively(Duration.ofSeconds(5), () -> {
            FibonacciHeap heap = new FibonacciHeap();
            for (int i = 1000; i >= 1; i--) {
                heap.insert(i);
            }
            for (int i = 1; i <= 1000; i++) {
                assertEquals(i, heap.min().key);
                heap.removeMin();
            }
        });
    }

    @Test
    void testShuffledInsert() {
        assertTimeoutPreemptively(Duration.ofSeconds(5), () -> {
            FibonacciHeap heap = new FibonacciHeap();
            List<Integer> nums = new ArrayList<>();
            for (int i = 1; i <= 10000; i++) {
                nums.add(i);
            }
        });
    }
}

```

```

    }
    Collections.shuffle(nums);

    for (int i : nums) {
        heap.insert(i);
    }

    for (int i = 1; i <= 10000; i++) {
        assertEquals(i, heap.min().key);
        heap.removeMin();
    }
});
}

@Test
void testDuplicates() {
    assertTimeoutPreemptively(Duration.ofSeconds(5), () -> {
        FibonacciHeap heap = new FibonacciHeap();
        int[] nums = {3, 3, 3, 4, 4, 4};

        for (int i : nums) {
            heap.insert(i);
        }

        for (int i : nums) {
            assertEquals(i, heap.min().key);
            heap.removeMin();
        }
    });
}

@Test
void testMerge() {
    assertTimeoutPreemptively(Duration.ofSeconds(5), () -> {
        FibonacciHeap h1 = new FibonacciHeap();
        FibonacciHeap h2 = new FibonacciHeap();

        h1.insert(5);
        h1.insert(1);
        h2.insert(4);
        h2.insert(2);

        h1.merge(h2);

        assertEquals(1, h1.min().key);

        int[] expected = {1, 2, 4, 5};
        for (int e : expected) {
            assertEquals(e, h1.min().key);
            h1.removeMin();
        }
    });
}

```

```

    });
}

@Test
void testDecreaseKeyNewMin() {
    assertTimeoutPreemptively(Duration.ofSeconds(5), () -> {
        FibonacciHeap heap = new FibonacciHeap();
        FibonacciHeap.Node n1 = heap.insert(10);
        heap.insert(20);

        heap.decreaseKey(n1, 1);
        assertEquals(1, heap.min().key);
    });
}

@Test
void testDecreaseKeyCascade() {
    assertTimeoutPreemptively(Duration.ofSeconds(5), () -> {
        FibonacciHeap heap = new FibonacciHeap();

        FibonacciHeap.Node a = heap.insert(10);
        FibonacciHeap.Node b = heap.insert(20);
        FibonacciHeap.Node c = heap.insert(30);

        heap.removeMin();
        heap.decreaseKey(c, 5);

        assertEquals(5, heap.min().key);
    });
}

@Test
void testStructureAfterOperations() {
    assertTimeoutPreemptively(Duration.ofSeconds(5), () -> {
        FibonacciHeap heap = new FibonacciHeap();
        heap.insert(3);
        heap.insert(1);
        heap.insert(2);

        FibonacciHeap.Node min = heap.min();
        assertEquals(1, min.key);

        heap.insert(0);
        assertEquals(0, heap.min().key);

        heap.removeMin();
        assertEquals(1, heap.min().key);
    });
}

@Test

```

```

void testMinOnEmptyHeap() {
    assertTimeoutPreemptively(Duration.ofSeconds(5), () -> {
        FibonacciHeap heap = new FibonacciHeap();
        assertNull(heap.min(), "min() on empty heap should return null");
    });
}

@Test
void testRemoveMinOnEmptyHeap() {
    assertTimeoutPreemptively(Duration.ofSeconds(5), () -> {
        FibonacciHeap heap = new FibonacciHeap();
        assertDoesNotThrow(heap::removeMin,
            "removeMin() on empty heap should not throw exception");
        assertNull(heap.min());
    });
}

@Test
void testSingleElementInsertRemove() {
    assertTimeoutPreemptively(Duration.ofSeconds(5), () -> {
        FibonacciHeap heap = new FibonacciHeap();
        heap.insert(42);

        assertEquals(42, heap.min().key);

        heap.removeMin();
        assertNull(heap.min(),
            "Heap should be empty after removing the only element");
    });
}

@Test
void testMergeWithEmptyHeap() {
    assertTimeoutPreemptively(Duration.ofSeconds(5), () -> {
        FibonacciHeap h1 = new FibonacciHeap();
        FibonacciHeap h2 = new FibonacciHeap();

        h1.insert(5);
        h1.insert(1);

        h1.merge(h2);
        assertEquals(1, h1.min().key);

        h2.merge(h1);
        assertEquals(1, h2.min().key);
    });
}

@Test
void testDecreaseKeyWithoutCut() {
    assertTimeoutPreemptively(Duration.ofSeconds(5), () -> {

```

```

        FibonacciHeap heap = new FibonacciHeap();
        FibonacciHeap.Node n1 = heap.insert(10);
        heap.insert(20);

        heap.decreaseKey(n1, 9);
        assertEquals(9, heap.min().key);
    });
}

@Test
void testReuseAfterFullClear() {
    assertTimeoutPreemptively(Duration.ofSeconds(5), () -> {
        FibonacciHeap heap = new FibonacciHeap();

        for (int i = 1; i <= 10; i++) {
            heap.insert(i);
        }
        for (int i = 1; i <= 10; i++) {
            heap.removeMin();
        }

        assertNull(heap.min(), "Heap should be empty");

        heap.insert(100);
        heap.insert(50);

        assertEquals(50, heap.min().key,
            "Heap should work correctly after being cleared");
    });
}
}

```

Результаты тестирования

```

-----
T E S T S
-----
Running org.example.heap.FibonacciTest
Tests run: 15, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.117 s - in org.example.heap.FibonacciTest
Results:
Tests run: 15, Failures: 0, Errors: 0, Skipped: 0

```

По результатам тестирования реализация Фибоначчиевой кучи корректно обрабатывает все предусмотренные сценарии, включая граничные случаи и работу с большими объёмами данных. Во всех тестах сохраняется свойство минимальной кучи, а извлечение элементов происходит в корректном порядке.

Структура корректно функционирует после различных комбинаций операций, включая вставку, удаление минимума, объединение куч и уменьшение ключа. Поведение на пустой куче и при наличии одного элемента соответствует ожидаемому.

Проверки с большим количеством элементов и ограничением по времени выполнения подтверждают устойчивость реализации и отсутствие деградации производительности в рамках протестированных сценариев.

Таким образом, тестирование подтверждает корректность и стабильность реализации в пределах охваченных тестами случаев.

Часть 3

Предметная область: после недолгой поездки на аэромобиле Артур и старый магратеянин оказались у какой-то двери. Они вышли из машины и прошли в приемную, уставленную стеклянными столиками и наградами из прозрачного пластика. Почти сразу же над дверью в другом конце комнаты загорелся свет, и они вошли в нее.

Доменная модель для заданного текста:

[ITMO/ТПО/Лаб1/Code/TPOLAB1/src/main/java/org/example/domain at master · AlexandroLe/ITMO](#)

Описание классов и их методов

1. Aeromobile - аэромобиль, средство перемещения персонажей

Aeromobile(String id) - создание аэромобиля с уникальным идентификатором.

getPassengers() - получение списка текущих пассажиров (только для чтения).

boardAllWithMarkers(List<Person>) - посадка группы персонажей в аэромобиль с удалением их из предыдущей локации.

approachDoor(Door) - приближение к двери (инициирует реакцию двери, например включение света).

disembarkAllWithMarkers(Location) - высадка всех пассажиров в указанную локацию.

toString() - строковое представление объекта.

2. Door - дверь между двумя локациями

Door(String name, Location a, Location b, Light light) - создание двери, соединяющей две локации, с источником света.

getName() - получение имени двери.

getLight() - получение связанного источника света.

lock() - блокировка двери.

unlock() - разблокировка двери.

isLocked() - проверка состояния блокировки.

otherSide(Location) - определение противоположной стороны двери относительно текущей локации.

approachWithMarkers() - реакция двери на приближение (включение света).

passThrough(Location) - переход через дверь при отсутствии блокировки.
toString() - строковое представление двери.

3. Item - предмет в локации

Item(String name, String material) - создание предмета с названием и материалом.
getName() - получение названия предмета.
getMaterial() - получение материала предмета.
toString() - строковое представление предмета.

4. Light - источник света

isOn() - проверка состояния света (включён/выключен).
turnOn() - включение света.
turnOff() - выключение света.

5. Location - место действия (пространство)

Location(String name, String description) - создание локации с названием и описанием.
getName() - получение названия локации.
getDescription() - получение описания локации.
addOccupant(Person) - добавление персонажа в локацию.
removeOccupant(Person) - удаление персонажа из локации.
getOccupants() - получение списка находящихся в локации персонажей.
getItems() - получение списка предметов в локации.
getDoors() - получение списка дверей, связанных с локацией.
addItem(Item) - добавление предмета в локацию.
addDoor(Door) - добавление двери к локации.
toString() - строковое представление локации.

6. Person - персонаж доменной модели

Person(String name) - создание персонажа без начальной локации.
Person(String name, Location initial) - создание персонажа с размещением в начальной локации.
getName() - получение имени персонажа.
getLocation() - получение текущей локации персонажа.
moveTo(Location) - перемещение персонажа в другую локацию с обновлением состояния обеих локаций.
inspectWithMarkers() - действие осмотра текущей локации.
toString() - строковое представление персонажа.

7. Main - демонстрация сценария

main(String[] args) - моделирование последовательности событий из текста: прибытие к двери, вход в приёмную, включение света и изменение состояний объектов.

Структура тестов

1. boardAndDisembarkMovesPassengersAndTurnsLightOn

Проверяет полный сценарий: посадка пассажиров в аэромобиль, приближение к двери (включение света), высадка в новую локацию и корректное обновление состояний объектов.

2. lockedDoorPreventsPassageAndApproachTurnsLightOn

Проверяет, что при заблокированной двери проход запрещён (выбрасывается исключение), но приближение к двери всё равно включает свет.

3. personMoveToUpdatesOccupantsLists

Проверяет корректность перемещения персонажа между локациями и обновление списков находящихся в них персонажей.

4. receptionContainsItemsAndDescription

Проверяет добавление предметов в локацию, отсутствие потери данных и корректность хранения описания.

5. repeatedBoardingDoesNotDuplicatePassengers

Проверяет, что повторная посадка одного и того же персонажа не приводит к его дублированию в списке пассажиров.

6. doorOtherSideIsSymmetric

Проверяет корректность определения противоположной стороны двери и симметричность перехода.

7. moveToNullThrows

Проверяет, что попытка перемещения персонажа в null-локацию приводит к выбросу исключения.

8. disembarkOnEmptyVehicleIsNoOp

Проверяет, что высадка из пустого аэромобиля не вызывает ошибок и не изменяет состояние локации.

9. addSameItemDoesNotDuplicate

Проверяет, что повторное добавление одного и того же предмета в локацию не приводит к его дублированию.

10. movingToSameLocationDoesNotDuplicateOccupant

Проверяет, что перемещение персонажа в ту же самую локацию не создаёт дубликатов в списке находящихся в ней.

11. approachIsIdempotentForLight

Проверяет, что повторное приближение к двери не нарушает состояние света (он остаётся включённым).

12. boardingRemovesFromOldLocation

Проверяет, что при посадке в аэромобиль персонаж корректно удаляется из предыдущей локации.

Тесты доменной модели

```
package org.example.domain;

import org.junit.jupiter.api.Test;

import java.util.Arrays;
import java.util.Collections;

import static org.junit.jupiter.api.Assertions.*;

class DomainTest {

    @Test
    void boardAndDisembarkMovesPassengersAndTurnsLightOn() {
        Location outside = new Location("outside", "near car");
        Location reception = new Location("reception", "glass tables");
        Light light = new Light();
        Door door = new Door("front", outside, reception, light);

        Person arthur = new Person("Arthur", outside);
        Person oldMag = new Person("OldMag", outside);
        Aeromobile aero = new Aeromobile("a1");

        assertTrue(outside.getOccupants().contains(arthur));
        assertTrue(outside.getOccupants().contains(oldMag));
        assertFalse(light.isOn());
        assertTrue(aero.getPassengers().isEmpty());

        aero.boardAllWithMarkers(Arrays.asList(arthur, oldMag));
        aero.approachDoor(door);
        aero.disembarkAllWithMarkers(reception);

        assertTrue(light.isOn(), "Light should be on after approach");
        assertTrue(reception.getOccupants().contains(arthur), "Arthur must be in reception");
        assertTrue(reception.getOccupants().contains(oldMag), "OldMag must be in reception");
        assertTrue(aero.getPassengers().isEmpty(), "Aeromobile should be empty after disembark");
        assertSame(reception, arthur.getLocation());
    }

    @Test
    void lockedDoorPreventsPassageAndApproachTurnsLightOn() {
        Location outside = new Location("outside", "");
        Location reception = new Location("reception", "");
        Light light = new Light();
        Door door = new Door("front", outside, reception, light);
        door.lock();
    }
}
```

```

    Person p = new Person("P", outside);

    door.approachWithMarkers();
    assertTrue(light.isOn(), "Light must be on after approach even if the door
is locked");

    IllegalStateException ex = assertThrows(IllegalStateException.class, () ->
door.passThrough(outside));
    assertTrue(ex.getMessage().toLowerCase().contains("lock"));
}

@Test
void personMoveToUpdatesOccupantsLists() {
    Location car = new Location("car", "in car");
    Location reception = new Location("reception", "room");
    Person p = new Person("Testy", car);

    assertTrue(car.getOccupants().contains(p));
    assertSame(car, p.getLocation());

    p.moveTo(reception);

    assertSame(reception, p.getLocation());
    assertFalse(car.getOccupants().contains(p));
    assertTrue(reception.getOccupants().contains(p));
}

@Test
void receptionContainsItemsAndDescription() {
    Location reception = new Location("reception", "glass tables and
trophies");
    Item table = new Item("glass table", "glass");
    Item trophy = new Item("trophy", "plastic");

    reception.addItem(table);
    reception.addItem(trophy);

    assertEquals(2, reception.getItems().size());
    assertTrue(reception.getItems().contains(table));
    assertTrue(reception.getItems().contains(trophy));
    assertEquals("glass tables and trophies", reception.getDescription());
}

@Test
void repeatedBoardingDoesNotDuplicatePassengers() {
    Location outside = new Location("outside", "");
    Person p = new Person("Solo", outside);
    Aeromobile aero = new Aeromobile("aX");

```

```

        aero.boardAllWithMarkers(Arrays.asList(p));
        aero.boardAllWithMarkers(Arrays.asList(p));

        assertEquals(1, aero.getPassengers().size(), "Passenger should not be
duplicated in vehicle");
    }

    @Test
    void doorOtherSideIsSymmetric() {
        Location l1 = new Location("L1", "");
        Location l2 = new Location("L2", "");
        Door d = new Door("d", l1, l2, new Light());

        assertSame(l2, d.otherSide(l1));
        assertSame(l1, d.otherSide(l2));
        assertNull(d.otherSide(new Location("other", "")));
    }

    @Test
    void moveToNullThrows() {
        Person p = new Person("p");
        assertThrows(IllegalArgumentException.class, () -> p.moveTo(null));
    }

    @Test
    void disembarkOnEmptyVehicleIsNoOp() {
        Location dest = new Location("dest", "");
        Aeromobile empty = new Aeromobile("empty");
        empty.disembarkAllWithMarkers(dest);
        assertTrue(dest.getOccupants().isEmpty());
    }

    @Test
    void addSameItemDoesNotDuplicate() {
        Location reception = new Location("reception", "");
        Item trophy = new Item("trophy", "plastic");
        reception.addItem(trophy);
        reception.addItem(trophy);
        assertEquals(1, reception.getItems().size());
    }

    @Test
    void movingToSameLocationDoesNotDuplicateOccupant() {
        Location room = new Location("room", "");
        Person p = new Person("X", room);
        p.moveTo(room);
        assertEquals(1, room.getOccupants().size());
    }

    @Test
    void approachIsIdempotentForLight() {

```

```

        Location outside = new Location("outside", "");
        Location reception = new Location("reception", "");
        Light light = new Light();
        Door door = new Door("d", outside, reception, light);
        door.approachWithMarkers();
        door.approachWithMarkers();
        assertTrue(light.isOn(), "light should remain on after repeated approach");
    }

    @Test
    void boardingRemovesFromOldLocation() {
        Location outside = new Location("outside", "");
        Location other = new Location("other", "");
        Person p = new Person("Z", other);
        Aeromobile aero = new Aeromobile("v");
        assertTrue(other.getOccupants().contains(p));
        aero.boardAllWithMarkers(Arrays.asList(p));
        assertFalse(other.getOccupants().contains(p), "boarding should remove
person from previous location");
        assertTrue(aero.getPassengers().contains(p));
    }
}

```

Результаты тестирования

```

[INFO] -----
[INFO]  T E S T S
[INFO] -----
[INFO] Running org.example.domain.DomainTest
|B| light:on
|b| |B| light:on
|b| Testy moved to reception
X moved to room
|B| light:on
|b| |A| boarded Z
|a| |C| |c| |A| boarded Arthur
boarded OldMag
|a| |B| light:on
|b| |C| Arthur moved to reception
OldMag moved to reception
|c| |A| boarded Solo
|a| |A| |a| [INFO] Tests run: 12, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.061 s - in org.example.domain.DomainTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 12, Failures: 0, Errors: 0, Skipped: 0
[INFO]

```

По результатам выполнения тестов доменной модели все проверки завершены успешно: выполнено 12 тестов, отказов и ошибок не зафиксировано, пропущенные тесты отсутствуют. Сборка проекта завершена со статусом BUILD SUCCESS.

Тестирование подтвердило корректность взаимодействия основных сущностей модели: персонажей, локаций, двери, источника света и аэромобиля. Проверены как штатные сценарии (посадка, перемещение, высадка, включение света, добавление предметов), так и

граничные случаи (пустой транспорт, заблокированная дверь, повторные операции, попытка перемещения в null).

Логи выполнения показывают ожидаемые переходы состояний (включение света, перемещение персонажей), что дополнительно подтверждает корректность последовательности операций.

Таким образом, реализация доменной модели демонстрирует согласованность поведения объектов, корректное обновление состояний и устойчивость к проверенным граничным ситуациям в рамках разработанного тестового покрытия.

Выводы

В ходе лабораторной работы мы реализовали и протестировали математическую функцию, структуру данных и простую доменную модель. Это позволило нам на практике применить разные подходы к тестированию.

При работе с функцией косинуса мы уделили внимание точности вычислений и проверке граничных значений. В задаче с кучей мы сосредоточились на проверке корректности операций и их поведения при разных входных данных, включая большие объёмы. В доменной модели основной акцент был сделан на корректности взаимодействия объектов и изменении их состояний.

В результате мы закрепили навыки написания модульных тестов, научились продумывать тестовые сценарии для обычных и граничных случаев и получили больше опыта в проверке как вычислительной логики, так и поведения объектов в системе.