Федеральное государственное автономное образовательное
учреждение высшего образования
Национальный исследовательский университет ИТМО

# Отчёт

# По лабораторной работе №1

## Вариант: 470407

Студенты:

*Долинный М. В.*

*Антипин Г. В.*

группа *P3332*

Преподаватель:

*Чупанов А. А.*

г. Санкт-Петербург, 2026 г.

# Описание задания

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

Введите вариант: 470407

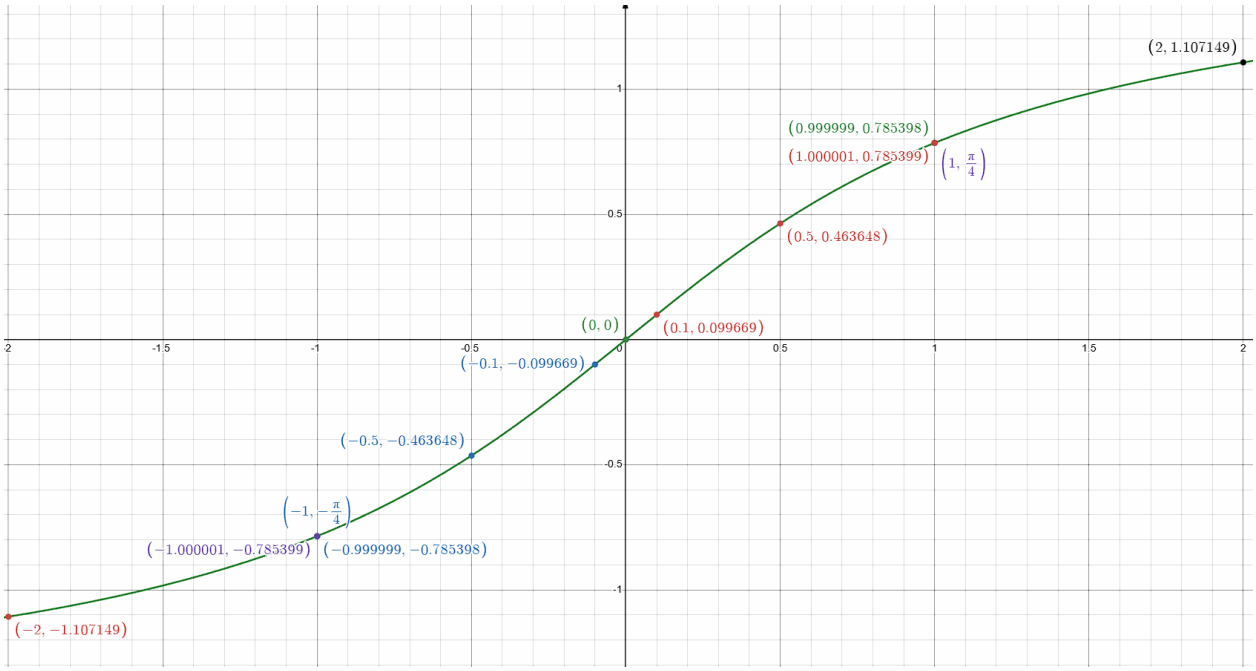
1. Функция arctg(x)
2. Программный модуль для работы с B+ деревьями (максимальное количество элементов в ключе - 6, http://www.cs.usfca.edu/~galles/visualization/BPlusTree.html)
3. Описание предметной области:

   Сквозь шум ветра раздался визг дудок, прямо из мостовой начали выскакивать горячие пончики по десять пенсов за штуку, из небес спикировала ужасная рыбина, и Артур с Фордом поняли, что нужно спасаться бегством.

## Выполнение

https://github.com/MikhailDolinnyi/tpo_lab1/tree/main

## **Задание 1:** *arctan(x)*




Тестовые точки выбраны таким образом, чтобы покрыть все значимые области и ветви алгоритма вычисления `arctan(x)`:

x = 0 - базовая точка для проверки корректности начальных условий.

|x| < 1 (±0.1, ±0.5, ±1e−10) - проверка сходимости и точности вычислений ряда Тейлора.

Граничные значения x = ±1 - проверка корректности обработки границ области сходимости.

Около границ (±0.9999999999, ±1.0000000001) — проверка условий перехода между ветвями алгоритма.

|x| > 1 (±2, 100) — проверка преобразования аргумента и корректности вычислений вне области сходимости.

NaN, ±∞ — проверка обработки некорректных входных данных.

Выбранный набор точек обеспечивает покрытие всех логических ветвей метода и граничных условий.

## Test Summary

| 19 | 0 | 0 | 0.102s |
|---|---|---|---|
| tests | failures | ignored | duration |

100% successful

Packages | Classes

| Package | Tests | Failures | Ignored | Duration | Success rate |
|---|---|---|---|---|---|
| ru.traphouse.utils | 19 | 0 | 0 | 0.102s | 100% |

task1

## task1

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ru.traphouse.utils | | 96 % | | 100 % | 1 | 8 | 1 | 15 | 1 | 2 | 0 | 1 |
| Total | 3 of 80 | 96 % | 0 of 12 | 100 % | 1 | 8 | 1 | 15 | 1 | 2 | 0 | 1 |

Функция **arctan(x)** корректно обрабатывает все тестовые случаи, включая граничные и специальные значения. Все результаты совпадают с эталонной реализацией **Math.atan**.

Функция правильно возвращает **NaN** для значений, выходящих за пределы допустимого диапазона, и сохраняет знак для нулевых значений.

Быстрая сходимость функции в окрестностях граничных значений не привела к ошибкам в вычислениях, что указывает на стабильность реализации.

**Задание 2:** *B+ tree*

# Описание модуля

В данной работе реализован программный модуль для работы с B+ деревом с максимальным количеством ключей в узле — 6.

**B+ дерево** — это сбалансированная структура данных, которая используется для эффективного хранения и поиска данных.

## В нашей реализации:

- все значения хранятся только в листьях;
- внутренние узлы используются для навигации по дереву;
- листья связаны между собой, что позволяет быстро проходить по всем элементам по порядку;
- при переполнении узла происходит его разбиение, а при необходимости — рост дерева вверх.
- Реализованы операции вставки и поиска, а также автоматическая балансировка дерева при переполнении узлов.

## Цель тестирования

**Цель тестирования** - проверить, что B+ дерево работает корректно в различных ситуациях.

В частности, необходимо убедиться, что:

- вставка элементов выполняется правильно;
- при переполнении листа или внутреннего узла дерево корректно делится и перестраивается;
- структура дерева остаётся сбалансированной;
- поиск находит существующие элементы и возвращает null для отсутствующих;
- при повторной вставке одного и того же ключа значение обновляется;
- алгоритм проходит через ожидаемые этапы работы (вставка в лист, split листа, split внутреннего узла, рост дерева).

Таким образом, тестирование направлено на проверку как правильности работы операций, так и корректности внутренней структуры B+ дерева после различных сценариев вставки.

## Test Summary

| 14 | 0 | 0 | 0.062s |
|---|---|---|---|
| tests | failures | ignored | duration |

**100%** successful

Packages | Classes

| Package | Tests | Failures | Ignored | Duration | Success rate |
|---|---|---|---|---|---|
| ru.traphouse.datastructure | 14 | 0 | 0 | 0.062s | 100% |

Generated by Gradle 8.14 at 17 февр. 2026 г., 20:45:08

task2

## task2

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ru.traphouse.datastructure | | 83 % | | 66 % | 9 | 31 | 21 | 92 | 5 | 19 | 0 | 5 |
| Total | 76 of 472 | 83 % | 8 of 24 | 66 % | 9 | 31 | 21 | 92 | 5 | 19 | 0 | 5 |

task2 > ru.traphouse.datastructure

## ru.traphouse.datastructure

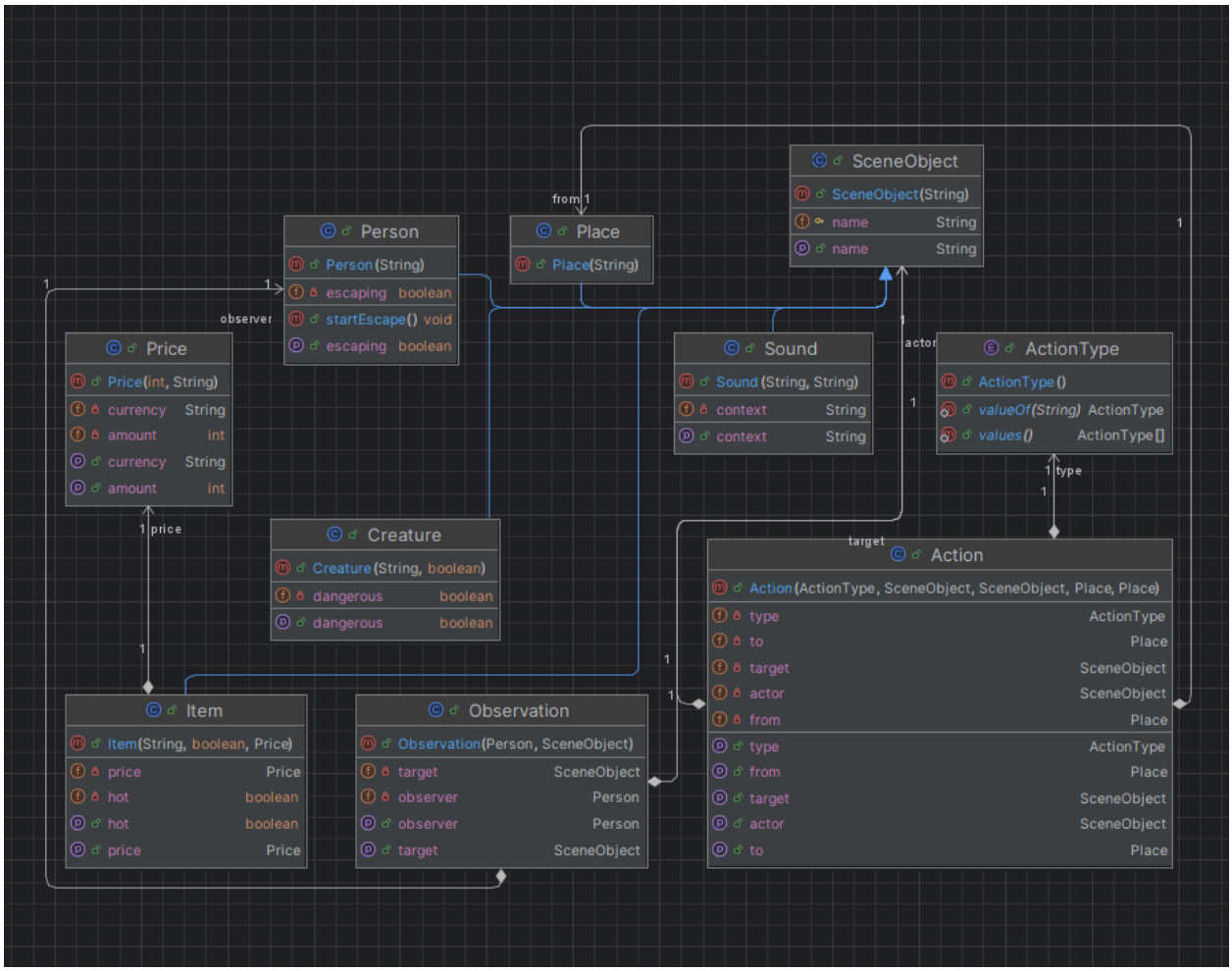
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BPlusTree | | 46 % | | 20 % | 7 | 12 | 19 | 34 | 3 | 7 | 0 | 1 |
| BPlusTree.InternalNode | | 98 % | | 100 % | 1 | 9 | 1 | 26 | 1 | 5 | 0 | 1 |
| BPlusTree.LeafNode | | 98 % | | 100 % | 1 | 8 | 1 | 26 | 1 | 5 | 0 | 1 |
| BPlusTree.Node | | 100 % | | n/a | 0 | 1 | 0 | 2 | 0 | 1 | 0 | 1 |
| BPlusTree.Split | | 100 % | | n/a | 0 | 1 | 0 | 4 | 0 | 1 | 0 | 1 |
| Total | 76 of 472 | 83 % | 8 of 24 | 66 % | 9 | 31 | 21 | 92 | 5 | 19 | 0 | 5 |

Результаты тестирования:

- B+ дерево реализовано корректно и проходит все тесты.
- Код покрыт модульными тестами.
- Тестирование подтвердило, что дерево корректно выполняет все основные операции и сохраняет свои свойства.

**Задание 3:** *Описание предметной области*

Для начала, спроектируем нашу доменную область в соответствии с заданным текстом.




# Описание предметной области

Предметная область моделирует сцену из текста как набор объектов и событий:

- SceneObject - базовая сущность сцены с именем (персонажи, предметы, места, звуки, существа).
- Person - персонаж (Артур, Форд) с состоянием *спасается/не спасается*.
- Place - место действия (мостовая, небеса), используется как источник/пункт назначения события.
- Sound - звуковое событие с контекстом (“визг дудок” на фоне “шума ветра”).
- Item + Price - предметы сцены (пончики) и их стоимость (10 пенсов).
- Creature - существо сцены (рыбина) с признаком опасности/ужасности.
- Action + ActionType - событие/действие в сцене (выскочило, спикировало, бегство) с указанием актора, цели и перемещения (from/to).
- Observation - фиксация того, что персонаж наблюдает объект/событие

Далее, в соответствии с TDD напишем тесты по спроектированной модели.

Цель тестирования — убедиться, что:

- Все команды выполняются корректно.
- Состояния персонажей и объектов изменяются в соответствии с командами.
- Сценарии выполняются последовательно и без ошибок.

Составим следующие тесты:

**PersonTest**

- storesName — имя персонажа сохраняется при создании.
- escapingStateChangesCorrectly — состояние escaping меняется после startEscape() и читается через isEscaping().

**PlaceTest**

- isSceneObjectAndStoresName — место хранит имя (наследование от SceneObject).

**SoundTest**

- storesNameAndContext — звук хранит имя события и контекст (“шум ветра”).

**PriceTest**

- storesAmountAndCurrency — цена хранит количество и валюту (“10 пенсов”).

**ItemTest**

- storesHotAndPrice — предмет хранит признак “горячий” и ссылку на цену.

**CreatureTest**

- storesDangerousFlag — существо хранит имя и признак опасности.

**ObservationTest**

- linksObserverAndTarget — наблюдение связывает наблюдателя (Person) и цель (SceneObject).

**ActionTest**

- storesAllFields — действие корректно сохраняет type/actor/target/from/to.
- scenarioFromTextIsRepresentable — проверяет, что события из текста можно выразить через объекты модели:
    - ERUPT (пончики появляются из мостовой),
    - DIVE (рыбина спикировала из небес к мостовой),
    - ESCAPE (Артур и Форд начали спасаться).

Как мы можем видеть, все работает корректно:

## Test Summary

| 10 | 0 | 0 | 0.048s |
|---|---|---|---|
| tests | failures | ignored | duration |

**100%** successful

Packages | Classes

| Package | Tests | Failures | Ignored | Duration | Success rate |
|---|---|---|---|---|---|
| ru.traphouse.model | 10 | 0 | 0 | 0.048s | 100% |

Generated by Gradle 8.14 at 17 февр. 2026 г., 21:01:10

task3 > ru.traphouse.model

## ru.traphouse.model

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Action | | 100 % | | n/a | 0 | 6 | 0 | 12 | 0 | 6 | 0 | 1 |
| ActionType | | 100 % | | n/a | 0 | 1 | 0 | 4 | 0 | 1 | 0 | 1 |
| Item | | 100 % | | n/a | 0 | 3 | 0 | 6 | 0 | 3 | 0 | 1 |
| Observation | | 100 % | | n/a | 0 | 3 | 0 | 6 | 0 | 3 | 0 | 1 |
| Price | | 100 % | | n/a | 0 | 3 | 0 | 6 | 0 | 3 | 0 | 1 |
| Person | | 100 % | | n/a | 0 | 3 | 0 | 6 | 0 | 3 | 0 | 1 |
| Sound | | 100 % | | n/a | 0 | 2 | 0 | 4 | 0 | 2 | 0 | 1 |
| Creature | | 100 % | | n/a | 0 | 2 | 0 | 4 | 0 | 2 | 0 | 1 |
| SceneObject | | 100 % | | n/a | 0 | 2 | 0 | 4 | 0 | 2 | 0 | 1 |
| Place | | 100 % | | n/a | 0 | 1 | 0 | 2 | 0 | 1 | 0 | 1 |
| Total | 0 of 147 | 100 % | 0 of 0 | n/a | 0 | 26 | 0 | 54 | 0 | 26 | 0 | 10 |

# Выводы

Во время выполнения лабораторной работы я углубил свои знания в JUnit5, научился писать юнит-тесты, использовать параметризированные тесты и тесты на проверку составленной объектной модели.