ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Дисциплина

«Тестирование программного обеспечения»

Лабораторная работа №1

Выполнил:

Студент группы P3331

Колмаков Дмитрий Владимирович

Преподаватель:

Гаврилов А.В.

Санкт-Петербург

2025 г.

Содержание

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 **31005**:

1. Функция arctg(x)
2. Программный модуль для работы с хеш-таблицей с закрытой адресацией (Hash Integer, http://www.cs.usfca.edu/~galles/visualization/OpenHash.html)
3. Описание предметной области:

   В полной темноте сверкнула ослепительно яркая точка света. Она начала расползаться в стороны, превращаясь в узкий полумесяц, и через несколько секунд показались два солнца: огненные светила, сжигающие белым пламенем черный край горизонта. Яркие цветные сполохи струились сквозь разреженную атмосферу.

Ход работы

Задание 1

В ходе первого задания я реализовал класс **ArctanSeries** и метод **arctan(double x, int n)**, реализующий функцию арктангенса с помощью разложения функции в ряд Тейлора из суммы **n** функций для параметра **x** по формуле:

$$\sum_{k=1}^{n} \frac{(-1)^k x^{2k+1}}{2k+1}.$$

При этом ряд сходится при $x \in [-1; 1]$.

Также реализовал класс с тестами **ArctanSeriesTest**, использующий фреймворк JUnit 4 для тестирования заданной функции. Были написаны методы для проверки корректности поведения функции в следующих случаях:

- **Базовые случаи** - проверяют корректную работу функции при x в границах сходимости и значениях n, обеспечивающих достаточную точность;
- **Нечетность функции** - проверяют изменение значения функции на противоположное при изменении знака x;
- **Монотонность функции** - проверяет, что большему значению аргумента соответствует большее значение функции;
- **Случайные аргументы** - проверяет корректность работы функции на 100 случайных значениях аргумента x;
- **Граничные случаи** - проверка граничных случаев, таких как:
    - Некорректные значения x (Double.POSITIVE_INFINITY, Double.NaN);
    - Значения x, которые выходят за границы;
    - Некорректные значения n ($n \leq 0$).

Таким образом, было проведено тестирование методом черного ящика, показавшее корректность реализации метода.

Задание 2

В ходе первого задания я реализовал класс **HashTable**, который представляет собой реализацию хэш-таблицы, использующую метод цепочек для разрешения коллизий. Основные поля и методы класса:

Поля:

- **size –** количество «корзин»;
- **table –** список связных списков, где каждый связный список хранит элементы, одной корзину.

Конструктор:

- **HashTable(int size)** — создает хэш-таблицу с заданным количеством корзин.

Методы:

- **private int hash(int key)**: Вычисляет индекс корзины для заданного ключа с помощью хэш-функции: **Math.abs(key) % size**;
- **public void insert(int key)**: Вставляет ключ в хэш-таблицу. Вычисляет индекс корзины с помощью хэш-функции и добавляет ключ в соответствующий связный список;
- **public boolean search(int key)**: Проверяет, содержится ли ключ в хэш-таблице;
- **public boolean delete(int key)**: Удаляет ключ из хэш-таблицы, если он присутствует;
- **public ArrayList<LinkedList<Integer>> getTable()**: Возвращает внутреннюю структуру хэш-таблицы (список связных списков). Используется для тестирования методом белого ящика.

Также реализовал класс с тестами **HashTableTest**, использующий фреймворк JUnit 4 для тестирования реализации хэш-таблицы. Были написаны методы для проверки корректности реализации в следующих случаях:

- **testInsert**: Проверяет вставку элементов в хэш-таблицу, включая случай коллизии (когда два ключа попадают в одну корзину);
- **testDelete**: Проверяет удаление элемента из хэш-таблицы;
- **testDeleteNonExistentElement**: Проверяет поведение при попытке удаления несуществующего элемента;
- **testInsertDuplicate**: Проверяет поведение при вставке дубликатов (одинаковых ключей).
- **testLargeNumberOfElements**: Проверяет корректность работы хэш-таблицы при большом количестве элементов.

Таким образом, было проведено тестирование методом белого ящика, показавшее корректность реализации хэш-таблицы.

Задание 3

В ходе третьего задания я сформировал доменную модель для заданного текста. Были реализованы классы **Atmosphere**, **Horizon**, **LightPoint**, **Moon**, **Sun**, представляющие собой объекты из заданного текста, и взаимосвязи между ними.

Также было разработано тестовое покрытие в классе **DomainModelTest**, использующее фреймворк JUnit 4 для тестирования данной доменной модели. Были протестированы отдельные методы каждого класса, а также поведение в соответствии с описанным в тексте сценарием.

Таким образом, было протестировано, что объекты доменной модели:

- Корректно создаются и инициализируются.
- Изменяют свое состояние в соответствии с сценарием.
- Взаимодействуют друг с другом так, как описано в тексте.
- Воспроизводят последовательность событий, соответствующую тексту.

Тесты подтверждают, что доменная модель корректно отражает описанную предметную область и поведение объектов.

.

Вывод

В ходе выполнения лабораторной работы были успешно выполнены три задания, каждое из которых было направлено на освоение различных аспектов разработки и тестирования программного обеспечения:

- **Задание 1**: Было проведено модульное тестирование методом черного ящика, которое охватило базовые, граничные и случайные случаи, а также проверку свойств функции (нечетность, монотонность). Тесты подтвердили корректность реализации функции и ее устойчивость к различным входным данным.
- **Задание 2:** Было проведено модульное тестирование методом белого ящика, которое включало проверку вставки, удаления, поиска элементов, а также обработку коллизий и дубликатов. Тесты подтвердили корректность работы хэш-таблицы и ее способность обрабатывать различные сценарии использования.
- **Задание 3:** Было проведено тестирование, которое подтвердило, что объекты модели корректно взаимодействуют друг с другом и воспроизводят описанный в тексте сценарий. Тесты охватили создание объектов, изменение их состояния и последовательность событий.