**Санкт-Петербургский Национальный Исследовательский Университет ИТМО**

**Лабораторная работа №1**

по дисциплине

**«Тестирование программного обеспечения»**

Студенты: Кенжаев Р.Е.

Гуменник П.О.

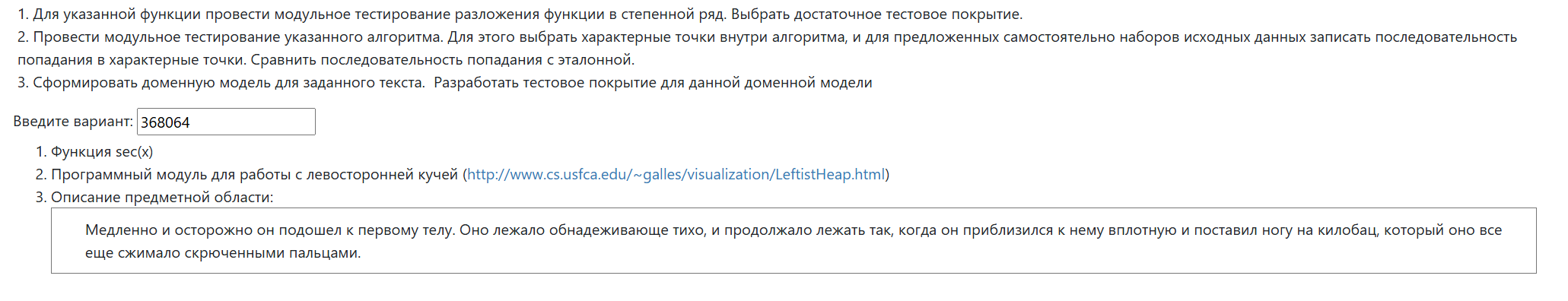
Группа: Р3333

Преподаватель: Инячина Д. А.

Санкт-Петербург

2025 г.

**Вариант:**



**Исходный код:**

<https://github.com/FreezinMoon/TPO_LAB1>

**Задание 1:**

По варианту дана функция секанса. Реализация: <https://github.com/FreezinMoon/TPO_LAB1/blob/master/src/main/java/task1/SecSeries.java>

Функция секанса реализована в классе SecSeries, в котором есть два метода: secByMath и secBySeries. Первый метод просто использует стандартную функцию Math.cos(x), чтобы вычислить секанс как 1/cos(x).

Второй метод (secBySeries) вычисляет секанс через ряд Маклорена. Он использует четные числа Эйлера, которые заранее заданы в массиве, и суммирует первые несколько членов ряда.

Для факториала знаменателя в коде есть отдельная вспомогательная функция factorial(n).

Реализация тестов:

<https://github.com/FreezinMoon/TPO_LAB1/blob/master/src/test/java/task1/SecSeriesTest.java>

Класс SecSeriesTest содержит тесты для проверки корректности вычисления секанса с использованием разложения в ряд Маклорена. Тесты написаны на JUnit 5 и сравнивают результаты secBySeries с эталонными значениями, полученными через встроенные математические функции. Допустимая погрешность тестов = 1e-6.

Сами тесты включают:

1. **testSecAtZero** – проверяет, что sec(0) = 1, как в ряде, так и в стандартной функции.
2. **testSecAtSmallValue** – сравнивает значения при малом аргументе x = 0.1.
3. **testSecAtNegativeValue** – аналогично предыдущему, но с отрицательным значением x = -0.2.
4. **testSecWithIncreasedTerms** – проверяет, что увеличение числа членов ряда улучшает точность вычислений.
5. **testSecNearBoundary** – тестирует поведение функции вблизи точки разрыва.
6. **testSecRandomValues** – выполняет массовое тестирование для случайных значений x, сравнивая результаты ряда с эталонным значением.

**Задание 2:**

Класс LeftistHeap реализует левостороннюю кучу – структуру данных для эффективного объединения двух куч.

Реализация:

<https://github.com/FreezinMoon/TPO_LAB1/blob/master/src/main/java/task2/LeftistHeap.java>

Куча поддерживает основные операции:

* **insert(int key)** – вставка нового элемента в кучу.
* **removeSmallest()** – удаление минимального элемента (корня кучи).
* **clearHeap()** – очистка всей кучи.
* **isEmpty()** – проверка, пуста ли куча.
* **getRootKey()** – получение минимального элемента без удаления.

Основной алгоритм работы основан на рекурсивном слиянии (merge), где при необходимости происходит перестановка поддеревьев и пересчет npl (null path length). Для анализа работы алгоритма добавлен механизм трассировки, который фиксирует ключевые этапы слияния (A, B, C, D, E, F).

Реализация тестов:

<https://github.com/FreezinMoon/TPO_LAB1/blob/master/src/test/java/task2/LeftistHeapTest.java>

В тестах проверяются основные операции с кучей, включая вставку, удаление минимума, очистку и поведение в граничных случаях.

**Основные тесты:**

1. **testInsertTrace()** – проверяет корректность трассировки при вставке элементов.
2. **testRemoveSmallestTrace()** – тестирует удаление минимального элемента и проверяет трассировку процесса.
3. **testClearHeapTrace()** – проверяет, что после очистки куча становится пустой и трассировка отсутствует.
4. **testInsertIncreasingAndRemove()** – вставка элементов в возрастающем порядке и их удаление, проверка правильности порядка.
5. **testInsertDecreasingAndRemove()** – аналогично предыдущему тесту, но с убывающим порядком вставки.
6. **testRemoveFromEmpty()** – проверяет, что удаление из пустой кучи вызывает исключение
7. **testClearTwice()** – проверяет, что повторный вызов очистки не вызывает ошибок.
8. **testRandomInsertAndRemove()** – вставляет 10 случайных чисел в кучу, удаляет их и сравнивает порядок с заранее отсортированным массивом.

**Задание 3:**

Дано описание предметной области:

**Медленно и осторожно он подошел к первому телу. Оно лежало обнадеживающе тихо, и продолжало лежать так, когда он приблизился к нему вплотную и поставил ногу на килобац, который оно все еще сжимало скрюченными пальцами.**

Реализация:

<https://github.com/FreezinMoon/TPO_LAB1/tree/master/src/main/java/task3>

Описание:

1. **Body** – представляет тело, которое может находиться в состоянии покоя или движения. Также оно может удерживать предмет.

* Позволяет получить и изменить текущее состояние тела.
* Проверяет, действительно ли оно держит предмет.

1. **BodyState** – перечисление возможных состояний тела:

* STILL – покой.
* MOVING – движение.

1. **Kilobac** – представляет предмет, который можно держать или на который можно наступить.

* Поддерживает изменение своего состояния (например, стал ли он под ногой персонажа).

1. **Person** – представляет персонажа, который может находиться близко (NEAR) или далеко (FAR) от тела.

* Может приблизиться к телу, но не может сделать это повторно (генерируется исключение).
* Может наступить на Kilobac, если уже находится рядом и тело не движется.

1. **PersonPosition** – перечисление возможных позиций персонажа:

* FAR – персонаж далеко.
* NEAR – персонаж рядом.

Классы моделируют простую систему взаимодействия тела, предмета (Kilobac) и персонажа, учитывая различные условия.

Реализация тестов:

<https://github.com/FreezinMoon/TPO_LAB1/blob/master/src/test/java/task3/PersonBodyKilobacTest.java>

Класс PersonBodyKilobacTest содержит **юнит-тесты** для проверки взаимодействия **персонажа (**Person**)**, **тела (**Body**)** и **предмета (**Kilobac**)**.

**Основные тесты:**

**testBasicScenario()**

* Тело (Body) неподвижно (STILL) и держит предмет (Kilobac).
* Персонаж (Person) приближается (FAR → NEAR).
* Персонаж ставит ногу на Kilobac, который остается в руках тела.

**testPlaceFootWhileFar()**

* Персонаж остается далеко (FAR).
* Попытка поставить ногу на Kilobac вызывает исключение.

**testBodyMoving()**

* Тело движется (MOVING).
* Персонаж рядом (NEAR), но попытка наступить на Kilobac запрещена (вызывает исключение).

**testKilobacFree()**

* Kilobac не удерживается телом.
* Персонаж приближается и без проблем придавливает Kilobac.

**testApproachTwice()**

* Персонаж приближается (FAR → NEAR).
* Повторная попытка приблизиться вызывает исключение.

**Вывод:**

В ходе выполнения лабораторной работы были реализованы и протестированы три различных задачи, каждая из которых имеет свою предметную область и логическую структуру.