

Лабораторная работа №3

Выполнила: Джумашева Камилла Исфатовна

Студент 6203-010302D группы

Ход выполнения

Задание 1 и 2

В ходе выполнения работы я изучила стандартные классы исключений Java. В пакете `functions` были разработаны два собственных класса исключений:

`FunctionPointIndexOutOfBoundsException` — наследуется от `IndexOutOfBoundsException`. Используется, когда мы пытаемся обратиться к точке по несуществующему индексу.

`InappropriateFunctionPointException` — наследуется от `Exception`. Служит для ситуаций, когда добавление или изменение точки нарушает логику табулированной функции (например, нарушение упорядоченности координат X).

Задание 3 и 6

Класс `TabulatedFunction` из прошлой работы был переименован в `ArrayTabulatedFunction`. Был выделен общий интерфейс `TabulatedFunction`, описывающий методы работы с функциями. В класс на массиве я добавила проверки аргументов методов. Теперь конструкторы выбрасывают `IllegalArgumentException`, если передать некорректные границы или недостаточное количество точек. Методы доступа (`getPoint`, `setPoint`) контролируют индексы, а методы модификации (`addPoint`, `setPointX`) следят за тем, чтобы значения X не нарушали сортировку массива, выбрасывая `InappropriateFunctionPointException`.

Задание 4

Наиболее трудоемкой частью работы стала реализация класса `LinkedListTabulatedFunction`, основанного на связном списке. В качестве структуры данных был выбран двусвязный циклический список с выделенной головой. Для этого внутри класса был создан статический вложенный класс `FunctionNode`, содержащий значение точки и ссылки на предыдущий (`prev`) и следующий (`next`) элементы. Использование статического класса обосновано тем, что узлу не требуется доступ к полям самого списка, он хранит только свои связи.

Были реализованы методы для манипулирования списком:

`addNodeToTail()` — добавляет элемент в конец (перед головой).

`deleteNodeByIndex()` — удаляет узел, перевязывая ссылки соседей друг на друга.

`getNodeByIndex(int index)` — возвращает ссылку на узел. Здесь была реализована оптимизация: если индекс находится в первой половине списка, поиск идет от головы (`head.next`), иначе — с конца (`head.prev`).

Задание 5

Методы интерфейса `TabulatedFunction` в классе связанного списка были реализованы с учетом особенностей структуры. Особое внимание было уделено методу `getFunctionValue()`. Вместо использования методов `getPoint()`, которые каждый раз искали бы элемент заново (что привело бы к квадратичной сложности), был реализован проход по списку через итерацию ссылок `current = current.next`. Это позволило вычислить значение функции за один проход ($O(N)$).

Задание 7

Для проверки работоспособности был модифицирован класс `Main`. Я создала объект `LinkedListTabulatedFunction` и провела серию тестов, намеренно вызывая ошибочные ситуации:

Попытка вставки точки с нарушением порядка X.

Попытка изменения координат, приводящая к коллизиям.

Удаление точек до недопустимого количества (< 3).

Обращение по неверному индексу.

Все исключения были корректно перехвачены блоками `try-catch`, что подтверждает надежность реализованной логики. Пример вывода консоли представлен ниже.

```

1. Создаем функцию и заполняем базовыми значениями ( $y = 2x$ )
Текущие точки: (0.0; 0.0) (5.0; 10.0) (10.0; 20.0) (15.0; 30.0) (20.0; 40.0)
Интервал X: от 0.0 до 20.0

2. Проверяем вычисление значений (интерполяцию):
При  $x = 0,0$ ,  $y = 0,0000$ 
При  $x = 0,5$ ,  $y = 1,0000$ 
При  $x = 1,0$ ,  $y = 2,0000$ 

3. Тестируем изменение точки (setPoint):
Точка с индексом 1 успешно заменена на (1, 2.005).
Текущие точки: (0.0; 0.0) (1.0; 2.005) (10.0; 20.0) (15.0; 30.0) (20.0; 40.0)
Пробуем вставить некорректную точку ( $x=-10$  на позицию 1)...
Все верно, поймали ошибку: нарушение порядка координат.

4. Тестируем изменение X (setPointX):
Координата X точки [1] успешно изменена на 4.
Текущие точки: (0.0; 0.0) (4.0; 2.005) (10.0; 20.0) (15.0; 30.0) (20.0; 40.0)
Пробуем задать  $x=600$  для точки [1] (должно быть запрещено)...
Система сработала верно, изменение отклонено.

5. Добавляем новую точку (addPoint):
Точка (15.5, 31) добавлена.
Текущие точки: (0.0; 0.0) (4.0; 2.005) (10.0; 20.0) (15.0; 30.0) (15.5; 31.0) (20.0; 40.0)
Пробуем добавить точку с уже существующим X (15.5)...
Верно, дубликаты X запрещены.

6. Удаляем точки (deletePoint):
Точка с индексом 1 удалена.
Текущие точки: (0.0; 0.0) (10.0; 20.0) (15.0; 30.0) (15.5; 31.0) (20.0; 40.0)
Пробуем удалить точку с индексом 100...
Поймали ошибку выхода за границы массива (как и ожидалось).

7. Тест на минимальное количество точек:
Сейчас точек: 5
Удаляем точки по одной, пока не сработает ограничение...
Удалили точку [0]. Осталось: 4
Удалили точку [0]. Осталось: 3
Удалили точку [0]. Осталось: 2
Стоп. Сработала защита: Number of points is less than 3

Тесты завершены.

```

Рис. 1

Заключение

В ходе работы тяжелее всего далась реализация цикличности списка. С первого раза не получилось правильно замкнуть head саму на себя в пустом списке, из-за чего возникал NullPointerException при добавлении первого элемента. Также пришлось повозиться с пересчетом ссылок в методе addNodeByIndex, чтобы не потерять связность цепочки prev и next. Однако использование фиктивной головы значительно упростило граничные условия (не нужно отдельно обрабатывать вставку в самое начало или конец).