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 связного 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.

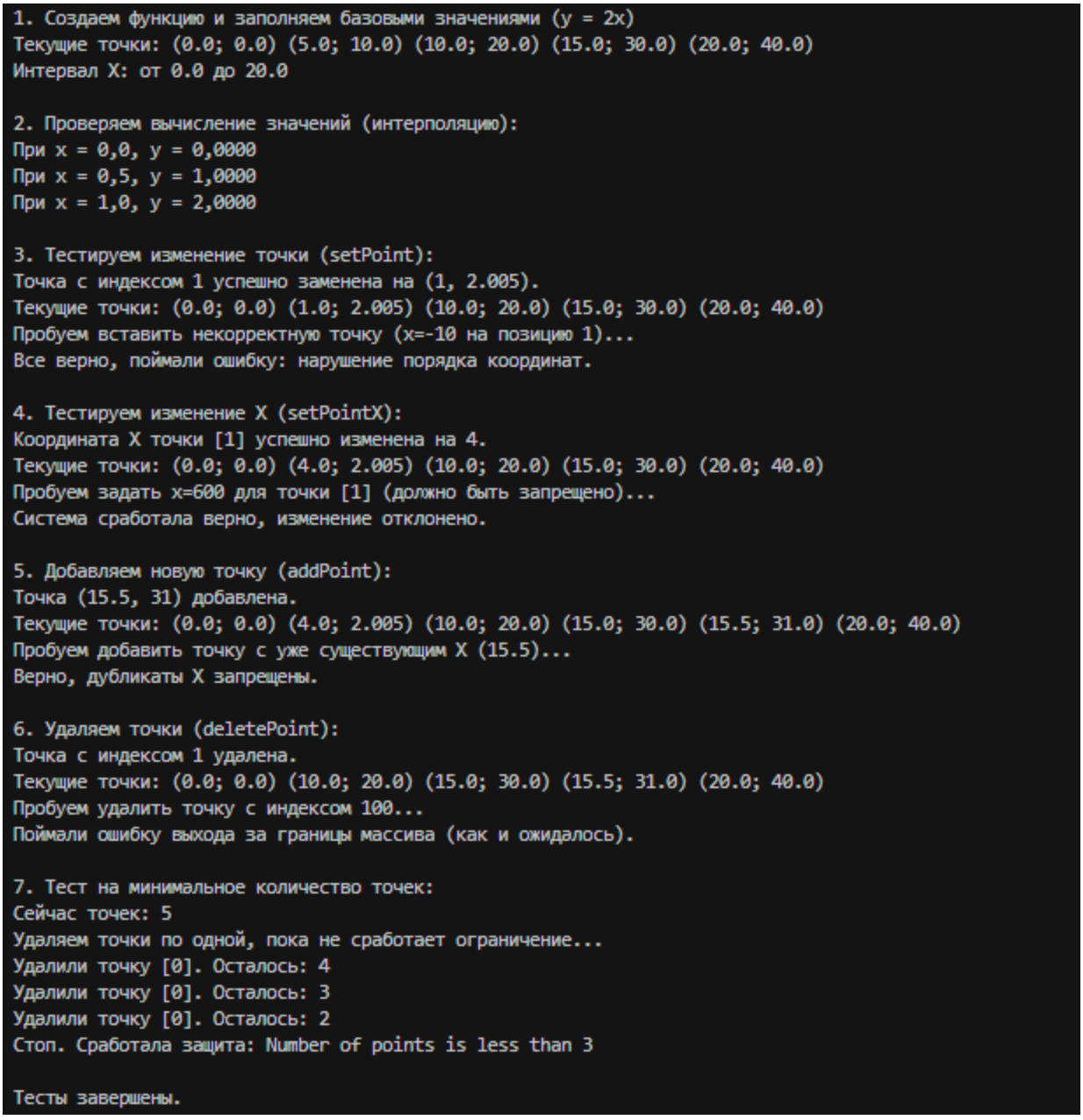

Рис. 1

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).