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.

```
 Тест аналитических функций и их табулирования
Сравнение значений:
x = 0,00 | Sin: 0,0000, TabSin: 0,0000 | Cos: 1,0000, TabCos: 1,0000
x = 0,50 | Sin: 0,4794, TabSin: 0,4721 | Cos: 0,8776, TabCos: 0,8646
x = 1,00 | Sin: 0,8415, TabSin: 0,8358 | Cos: 0,5403, TabCos: 0,5360
x = 1,50 | Sin: 0,9975, TabSin: 0,9848 | Cos: 0,0707, TabCos: 0,0704
x = 2,00 | Sin: 0,9093, TabSin: 0,8981 | Cos: -0,4161, TabCos: -0,4117
x = 2,50 | Sin: 0,5985, TabSin: 0,5941 | Cos: -0,8011, TabCos: -0,7942
x = 3,00 | Sin: 0,1411, TabSin: 0,1387 | Cos: -0,9900, TabCos: -0,9755

 Тест операций (Sin^2 + Cos^2)
x = 0,00: sin^2 + cos^2 = 1,0000
x = 0,50: sin^2 + cos^2 = 0,9704
x = 1,00: sin^2 + cos^2 = 0,9859
x = 1,50: sin^2 + cos^2 = 0,9748
x = 2,00: sin^2 + cos^2 = 0,9762
x = 2,50: sin^2 + cos^2 = 0,9836
x = 3,00: sin^2 + cos^2 = 0,9709

 Тест символьного ввода/вывода (Exp)
Функция записана в файл exp_char.txt
Функция считана из файла. Сравнение значений:
x = 0,0: Исходная = 1,0000, Считанная = 1,0000
x = 2,0: Исходная = 7,3891, Считанная = 7,3891
x = 4,0: Исходная = 54,5982, Считанная = 54,5982
x = 6,0: Исходная = 403,4288, Считанная = 403,4288
x = 8,0: Исходная = 2980,9580, Считанная = 2980,9580
x = 10,0: Исходная = 22026,4658, Считанная = 22026,4658

 Тест байтового ввода/вывода (Log)
Функция записана в файл log_byte.bin
Функция считана из файла. Сравнение значений:
x = 1,0: Исходная = -0,1310, Считанная = -0,1310
x = 3,0: Исходная = 1,0942, Считанная = 1,0942
x = 5,0: Исходная = 1,6084, Считанная = 1,6084
x = 7,0: Исходная = 1,9456, Считанная = 1,9456
x = 9,0: Исходная = 2,1972, Считанная = 2,1972

 Тест сериализации (Externalizable: Log(Exp(x)))
Объект (Externalizable) сериализован в файл complex_ser.bin
Объект десериализован. Проверка значений:
x = 0,0: Значение = 0,0000
x = 2,0: Значение = 2,0000
x = 4,0: Значение = 4,0000
x = 6,0: Значение = 6,0000
x = 8,0: Значение = 8,0000
x = 10,0: Значение = 10,0000
```

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).