Лабораторная работа №3

Выполнил: Мурзяков Егор 6204 – 010302D

Ход выполнения задания № 2

Согласно заданию в пакете “functions” были созданы два класса исключений для обработки особых ситуаций при работе с табулированными функциями.

Был создан класс “FunctionPointIndexOutOfBoundsException”, который наследуется от стандартного класса “IndexOutOfBoundsException”. Данное исключение предназначено для обработки случаев выхода за границы набора точек при обращении к ним по номеру.

Также был создан класс “InappropriateFunctionPointException”, наследуемый от класса “Exception”. Это исключение выбрасывается при попытке добавления или изменения точки функции несоответствующим образом.

Для обоих классов были реализованы конструкторы по умолчанию и конструкторы, принимающие текстовое сообщение об ошибке. Для класса “InappropriateFunctionPointException” дополнительно был реализован конструктор, принимающий причину исключения.

Ход выполнения задания № 3

В разработанный ранее класс “ArrayTabulatedFunction” были внесены изменения, обеспечивающие выбрасывание исключений методами класса в соответствии с заданием.

В конструкторах класса была добавлена проверка параметров на корректность. Оба конструктора теперь выбрасывают исключение “IllegalArgumentException”, если левая граница области определения больше или равна правой, а также если предлагаемое количество точек меньше двух. Это гарантирует, что объект функции будет создан только при корректных входных параметрах.

Для методов работы с точками функции были добавлены проверки выхода за границы набора точек. Методы “getPoint()”, “setPoint()”, “getPointX()”, “setPointX()”, “getPointY()”, “setPointY()” и “deletePoint()” теперь выбрасывают исключение “FunctionPointIndexOutOfBoundsException”, если переданный индекс выходит за допустимые пределы.

Была реализована проверка сохранения упорядоченности точек по оси абсцисс. Методы “setPoint()” и “setPointX()” выбрасывают исключение “InappropriateFunctionPointException”, если координата X задаваемой точки выходит за интервал, определяемый соседними точками. Аналогично, метод “addPoint()” проверяет, что добавляемая точка имеет уникальную абсциссу, не совпадающую с уже существующими точками функции.

Метод “deletePoint()” дополнительно проверяет, что после удаления точки в наборе останется не менее двух точек. В случае попытки удаления точки при количестве точек менее трех выбрасывается исключение “IllegalStateException”.

Ход выполнения задания № 4

В пакете functions был создан класс “LinkedListTabulatedFunction”, объект которого описывает табулированную функцию с использованием динамической структуры данных - связного списка. Данная реализация отличается от предыдущей тем, что вместо массива для хранения точек используется двусвязный циклический список с выделенной головой.

Для реализации связного списка был описан внутренний класс “FunctionNode”, представляющий элемент списка. Класс был сделан приватным и статическим, что обеспечивает правильную инкапсуляцию. Каждый узел содержит информационное поле типа “FunctionPoint” для хранения данных точки, а также ссылки на предыдущий и следующий элементы.

В основном классе “LinkedListTabulatedFunction” были объявлены следующие поля:

- “head” - ссылка на голову списка, которая всегда присутствует и не содержит данных

- “pointsCount” - количество точек в функции, что позволяет избежать подсчета длины списка каждый раз

- “lastUsedPoint” и “indexLastUsedPoint” - вспомогательные поля для оптимизации доступа к элементам

Были реализованы основные методы для работы со списком:

- “getNodeByIndex(int index)” - возвращает узел по индексу с оптимизацией доступа через выбор направления обхода от головы или последнего использованного узла

- “addNodeToTail()” - добавляет новый узел в конец списка

- “addNodeByIndex(int index)” - вставляет узел в указанную позицию

- “deleteNodeByIndex(int index)” - удаляет узел по индексу

Конструкторы класса создают табулированную функцию с равномерным распределением точек на заданном интервале. Также был реализован приватный конструктор для создания пустого списка, который может использоваться для последующего наполнения точками.

Ход выполнения задания № 5

В соответствии с заданием были реализованы конструкторы и методы, полностью аналогичные по параметрам и сигнатурам соответствующим конструкторам и методам класса “ArrayTabulatedFunction”.

В отличие от реализации на массивах, где необходимо каждый раз обращаться к элементам через индекс, в связном списке мы можем напрямую манипулировать ссылками между узлами. Это позволило значительно ускорить выполнение операций вставки и удаления точек.

Были реализованы все необходимые проверки и исключения, полностью соответствующие таковым в классе “ArrayTabulatedFunction”.

Ход выполнения задания № 6

Класс “TabulatedFunction” был переименован в “ArrayTabulatedFunction”.

Был создан интерфейс “TabulatedFunction”, содержащий объявления всех общих методов, которые должны поддерживать различные реализации табулированных функций.

Все методы в интерфейсе объявлены с теми же сигнатурами и исключениями, что и в исходных классах, что гарантирует полную совместимость существующего кода с новой архитектурой. Теперь суть работы с табулированными функциями заключена в типе интерфейса, а конкретные классы содержат только детали реализации.

Ход выполнения задания № 7

Для проверки корректности работы написанных классов был изменен ранее созданный класс “Main”, содержащий точку входа в программу. Тестирование проводилось для обеих реализаций табулированной функции - “ArrayTabulatedFunction” и “LinkedListTabulatedFunction”.

Для обеспечения полиморфизма ссылочная переменная для работы с объектом функции объявлялась типа интерфейса “TabulatedFunction”, а при создании объекта указывался конкретный класс реализации.

Проверим класс “ArrayTabulatedFunction”, для начала выведем значения функции (Рис. 1), затем протестируем корректные операции с точками (Рис. 2)

```
Начальные значения функции:
x:      y:
-3,000      11,000
-2,500      8,250
-2,000      6,000
-1,500      4,250
-1,000      3,000
-0,500      2,250
0,000       2,000
0,500       2,250
1,000       3,000
1,500       4,250
2,000       6,000
2,500       8,250
3,000       11,000
3,500       14,250
4,000       18,000
Область определения: [-3.0, 4.0]
Количество точек: 15
```

Рис. 1

```
Операции с точками:
Успешно изменили точку 0
Успешно изменили точку 5
Успешно добавили точку (2.7, 8.35)
x:      y:
-2,900      5,000
-2,500      8,250
-2,000      6,000
-1,500      4,250
-1,000      3,000
-0,500      2,400
0,000       2,000
0,500       2,250
1,000       3,000
1,500       4,250
2,000       6,000
2,500       8,250
2,700       8,350
3,000       11,000
3,500       14,250
4,000       18,000
Область определения: [-2.9, 4.0]
Количество точек: 16
```

Рис. 2

Проверим метод “getFunctionValue” (Рис. 3)

```
Значение функции в различных точках:
f(-2.5) = 8.25
f(0) = 2.0
f(2.3) = 7.35
f(2.5) = 8.25
f(2.0) = 6.0
f(2.8) = 9.233333333333333
f(3) = 11.0
f(5) = NaN
```

Рис. 3

Попробуем удалить точку с индексом 2 (Рис. 4)

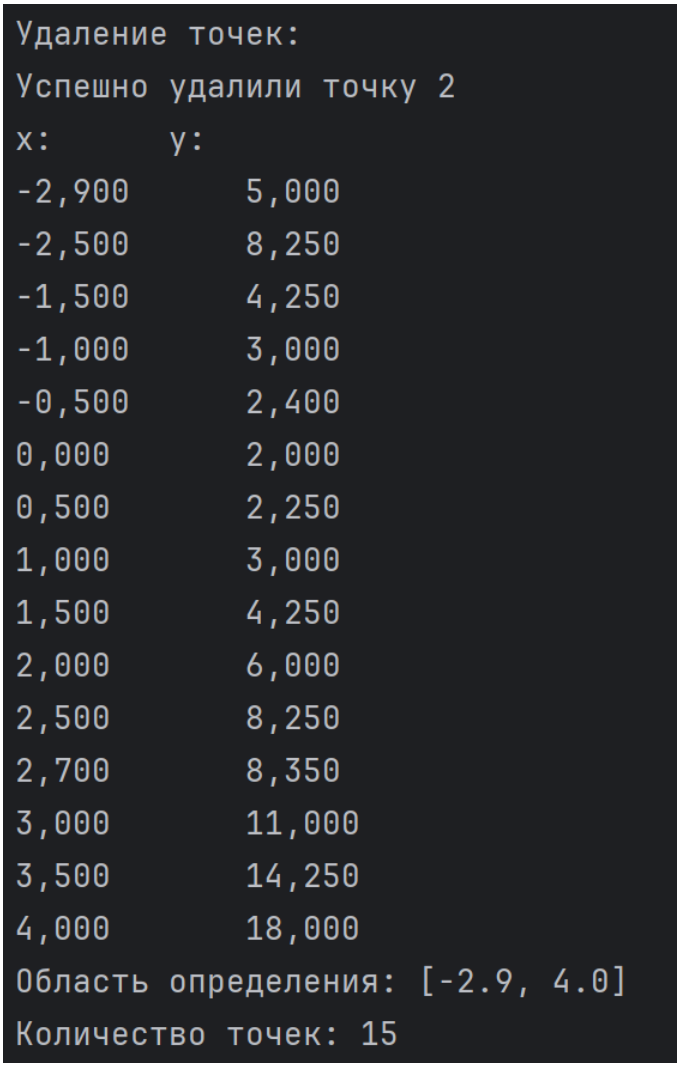

Рис. 4

Далее проверим класс “LinkedListTabulatedFunction” с той же функцией и теми же границами, значения функций (Рис. 5)

```
=== Тестирование LinkedListTabulatedFunction ===
Начальные значения функции:
x:      y:
-3,000      11,000
-2,500      8,250
-2,000      6,000
-1,500      4,250
-1,000      3,000
-0,500      2,250
0,000       2,000
0,500       2,250
1,000       3,000
1,500       4,250
2,000       6,000
2,500       8,250
3,000       11,000
3,500       14,250
4,000       18,000
Область определения: [-3.0, 4.0]
Количество точек: 15
```

Рис. 5

Операции с точками (Рис. 6)

```
Операции с точками:
Успешно изменили точку 0
Успешно изменили точку 5
Успешно добавили точку (2.7, 8.35)
x:      y:
-2,900      5,000
-2,500      8,250
-2,000      6,000
-1,500      4,250
-1,000      3,000
-0,500      2,400
0,000       2,000
0,500       2,250
1,000       3,000
1,500       4,250
2,000       6,000
2,500       8,250
2,700       8,350
3,000       11,000
3,500       14,250
4,000       18,000
Область определения: [-2.9, 4.0]
Количество точек: 16
```

Рис. 6

Проверим метод “getFunctionValue” (Рис. 7)

```
Значение функции в различных точках:
f(-2.5) = 8.25
f(0) = 2.0
f(2.3) = 7.35
f(2.5) = 8.25
f(2.0) = 6.0
f(2.8) = 9.23333333333333
f(3) = 11.0
f(5) = NaN
```

Рис. 7

Удалим точку (Рис. 8)

```
Удаление точек:
Успешно удалили точку 2
x:      y:
-2,900      5,000
-2,500      8,250
-1,500      4,250
-1,000      3,000
-0,500      2,400
0,000       2,000
0,500       2,250
1,000       3,000
1,500       4,250
2,000       6,000
2,500       8,250
2,700       8,350
3,000       11,000
3,500       14,250
4,000       18,000
Область определения: [-2.9, 4.0]
Количество точек: 15
```

Рис. 8

Были протестированы все виды исключений, предусмотренные в классах:

- “IllegalArgumentException” при некорректных параметрах конструкторов (Рис. 9)

```
1. Тестирование исключений при создании:
Поймано исключение: Левая граница области определения должна быть строго меньше правой
Поймано исключение: Количество точек должно быть больше или равно двум
```

Рис. 9

- “FunctionPointIndexOutOfBoundsException” при обращении к несуществующим индексам точек (Рис. 10)

```
Поймано исключение при getPoint(-1): Переданный номер выходит за границы набора точек
Поймано исключение при getPoint(5): Переданный номер выходит за границы набора точек
```

Рис. 10

- “InappropriateFunctionPointException” при нарушении порядка точек или добавлении точки с существующей абсциссой (Рис. 11)

```
Поймано исключение при setPoint: Координата x заданной точки должна быть меньше следующей
Поймано исключение при addPoint: Координата x новой точки совпадает с абсциссой уже существующей точки
```

Рис. 11

- “IllegalStateException” при попытке удалить точку, если после удаления останется менее двух точек (Рис. 12)

```
3. Тестирование исключений при удалении:
Поймано исключение при deletePoint: Точек в наборе не должно остаться менее двух
```

Рис. 12

Тестирование показало, что обе реализации работают корректно и выдают идентичные результаты для одинаковых входных данных. Все исключения выбрасываются корректно с соответствующими сообщениями об ошибках.