

## **Лабораторная работа №3**

Автор: Терин Ярослав

Группа: 6203-010302D

## **Задание 2**

Создали два новых класса “FunctionPointIndexOutOfBoundsException” и “InappropriateFunctionPointException” внутри пакета “functions”. Эти классы должны сообщать о 1-й класс наследует от класса “IndexOutOfBoundsException” и имеет конструктор родительского класса, который передаёт сообщение об ошибке который мы вызываем через ключевое слово супер. Так-же есть конструктор по умолчанию который создает исключение без параметров.

## **Задание 3**

В этом задание нужно было реализовать отдельные исключения для каждого из методов. Для каждого метода где исключение относилось к двум созданным нами классам было написано “throw” для всех исключений класса входящий в API нужно было прописывать “new”. Для методов “setPoint” и “setPointX” была добавлена проверка упорядоченности точек по оси X - если новая координата x не находилась строго между соседними точками, выбрасывалось исключение “InappropriateFunctionPointException”. Аналогично, метод “addPoint” проверял отсутствие дублирования координат x в существующем наборе точек. Метод “deletePoint” был дополнен проверкой минимального количества точек - при попытке удалить точку, когда в массиве оставалось менее трех точек, выбрасывалось исключение “IllegalStateException”.

## Задание 4

В этом задании был создан класс “`LinkedListTabulatedFunction`”, описывающий табулированную функцию с использованием связного списка. Внутри этого класса был реализован внутренний класс “`FunctionNode`”, содержащий три приватных поля: “`point`” для хранения объекта “`FunctionPoint`”, “`next`” и “`prev`” для хранения ссылок на следующий и предыдущий узлы. В классе “`FunctionNode`” были созданы два конструктора - первый принимал точку “`FunctionPoint`” и инициализировал поля класса, второй создавал объект с “`null`” значениями. Также я добавил геттеры и сеттеры для всех полей, чтобы обеспечить сохранность данных. Во внешнем классе “`LinkedListTabulatedFunction`” были объявлены приватные поля: “`head`” типа “`FunctionNode`”, который служил головой списка, и “`size`” для хранения количества элементов. Был реализован метод “`getNodeByIndex`”, возвращающий ссылку на узел по заданному индексу. Для быстроты доступа добавлялась проверка положения индекса в списке - если индекс находился в первой половине, поиск начинался с головы списка с последовательным переходом по “`next`”, если во второй - с конца списка с переходом по “`prev`”. Метод “`addNodeToTail`” добавлял новый узел в конец списка. Создавался новый узел, устанавливались связи с предыдущим последним элементом и головой списка, обновлялся счетчик “`size`” и возвращалась ссылка на созданный узел. Метод “`addNodeByIndex`” вставлял узел в указанную позицию. Если индекс равнялся “`size`”, использовался “`addNodeToTail`”, иначе находился узел в позиции “`index`” через “`getNodeByIndex`”, создавался новый узел и корректировались связи соседних элементов. Метод “`deleteNodeByIndex`” удалял узел по заданному индексу. Находился удаляемый узел через “`getNodeByIndex`”, изменялись связи его соседей так, чтобы они ссылались друг на друга, обнулялись связи удаляемого узла и уменьшался “`size`”. Метод возвращал ссылку на удаленный узел.

## **Задание 5**

В этом задании в класс “`LinkedListTabulatedFunction`” были реализованы конструкторы и методы для работы с табулированной функцией, аналогичные классу “`ArrayTabulatedFunction`”. Конструкторы принимали границы области определения вместе с количеством точек или массивом значений с выполнением проверок на корректность данных и выбрасыванием “`IllegalArgumentException`” при нарушении условий. Все методы интерфейса “`TabulatedFunction`”, включая получение границ области определения, вычисление значения функции через линейную интерполяцию, работу с отдельными точками, были реализованы через систему кэширования. Методы “`getLeftDomainBorder`” и “`getRightDomainBorder`” использовали прямое обращение к крайним узлам через “`head.getNext`” и “`head.getPrev`”, что позволяло мгновенно получать границы области определения без поиска по списку. При пустом списке возвращался “`Double.NaN`”. В методе “`getFunctionValue`” мы последовательно перебирали узлы списка через “`getNodeByIndex`” для нахождения интервала интерполяции, методы работы с точками использовали быстрый доступ через “`lastAccessedNode`”, который обновлялся при каждом обращении к “`getNodeByIndex`”. Еще при добавлении новых точек в методе “`addPoint`” сделали проверку на уникальность X через сравнение со всеми существующими точками, а затем находили позицию вставки путем последовательного сравнения его.

## **Задание 6**

В этом задании переименовали класс “TabulatedFunction”. Сам класс реализовали как интерфэйс описывающий методы которые должны присутствовать у любой табличной функции, независимо от способа её внутренней реализации. Так-же дописали в класс “LinkedListTabulatedFunction” и “ArrayTabulatedFunction” ключевое слово “implements” чтобы дать понять что мы их используем.

## Задание 7

Создал новые таб. функции для тестирования и проверил старые, но уже инициализировал их как списки. Вывод полученных исключений показан на фото снизу.

```
Создали функцию через первый конструктор (y=5x)
X: 1.0 Y: 5.0
X: 2.0 Y: 10.0
X: 3.0 Y: 15.0
X: 4.0 Y: 20.0
X: 5.0 Y: 25.0
X: 6.0 Y: 30.0
X: 7.0 Y: 35.0
X: 8.0 Y: 40.0
X: 9.0 Y: 45.0
X: 10.0 Y: 50.0

Успешно изменили X точки с индексом 2 на 3.5
Ошибка при изменении X точки на 100: X=100.0 должен быть строго между 1.0 и 3.5
Успешно заменили точку с индексом 4

Проверка второго конструктора
X: 1.0 Y: 4.4
X: 1.6666666666666665 Y: 5.5
X: 2.333333333333333 Y: 6.6
X: 3.0 Y: 7.7
X: 3.6666666666666665 Y: 8.8
X: 4.333333333333333 Y: 9.9
X: 5.0 Y: 10.0

Левая границы 5.0 >= правой границы 0.0
Индекс 100 выходит за границы [0, 5]
Точка с X=1.0 уже существует
X: 1.0 Y: 1.0
X: 1.4 Y: 3.0
X: 1.8 Y: 2.0
X: 2.2 Y: 4.0
X: 2.6 Y: 5.0
X: 3.0 Y: 6.0
Exception in thread "main" java.lang.IllegalArgumentException: Create breakpoint : Левая граница 4.0 >= правой границы 3.0
        at functions.LinkedListTabulatedFunction.<init>(LinkedListTabulatedFunction.java:74)
        at Main.main(Main.java:128)
Process finished with exit code 1

22.5 Функция определения значения в произвольной точке 4.5

Добавили y=80 по индексу 3
X: 1.0 Y: 5.0
X: 2.0 Y: 10.0
X: 3.0 Y: 15.0
X: 4.0 Y: 80.0
X: 5.0 Y: 25.0
X: 6.0 Y: 30.0
X: 7.0 Y: 35.0
X: 8.0 Y: 40.0
X: 9.0 Y: 45.0
X: 10.0 Y: 50.0

Добавили точку test_point 8.5, 17
X: 1.0 Y: 5.0
X: 2.0 Y: 10.0
X: 3.0 Y: 15.0
X: 4.0 Y: 80.0
X: 5.0 Y: 25.0
X: 6.0 Y: 30.0
X: 7.0 Y: 35.0
X: 8.0 Y: 40.0
X: 8.5 Y: 17.0
X: 9.0 Y: 45.0
X: 10.0 Y: 50.0

Удалили элемент с индексом 9
X: 1.0 Y: 5.0
X: 2.0 Y: 10.0
X: 3.0 Y: 15.0
X: 4.0 Y: 80.0
X: 5.0 Y: 25.0
X: 6.0 Y: 30.0
X: 7.0 Y: 35.0
X: 8.0 Y: 40.0
X: 8.5 Y: 17.0
X: 10.0 Y: 50.0
```