

## **Лабораторная работа №2**

Выполнил: Мурзяков Егор 6204 – 010302D

## Ход выполнения задания № 2

Согласно заданию в пакете “functions”, созданном ранее, был создан класс “FunctionPoint”, объект которого описывает одну точку табулированной функции. Для этого были созданы приватные поля, содержащие координаты точки по оси абсцисс и ординат. Для чтения и записи этих полей были созданы публичные методы: “getX”, “getY”, “setX”, “setY”. Далее были описаны следующие конструкторы:

- 1) “FunctionPoint(double x, double y)” создает объект точки с заданными координатами.
- 2) “FunctionPoint(FunctionPoint point)” создает объект точки с теми же координатами, что и у указанной точки
- 3) “FunctionPoint()” создаёт точку с координатами (0; 0)

### Ход выполнения задания № 3

Следуя заданию, в пакете “functions” был создан класс “TabulatedFunction”, объект которого описывает табулированную функцию. Были созданы приватные поля “points” и “pointsCount”, где поле “points” является массивом типа “FunctionPoint”.

Далее были описаны следующие конструкторы:

- 1) “TabulatedFunction(double leftX, double rightX, int pointsCount)” создаёт объект табулированной функции по заданным левой и правой границе области определения, а также количеству точек для табулирования. Сначала вычисляется шаг, который будет равен разнице границ, поделенной на количество промежутков. Количество промежутков на единицу меньше количества точек, которое нам известно. Проходим в цикле от крайней левой точки до крайней правой с шагом, который мы только что вычислили и заполняем массив “points” объектами класса “FunctionPoint”, для этого используем конструктор, который принимает на вход координаты точки, тк значение функции на данном этапе нам не известно, будем задавать ей значение = 0.
- 2) “TabulatedFunction(double leftX, double rightX, double[] values)” аналогичен предыдущему конструктору, но вместо количества точек получает значения функции в виде массива. В данном случае количество точек можно узнать через поле “length”, а значение функции дано нам в массиве “values”.

#### **Ход выполнения задания № 4**

В классе “TabulatedFunction” были описаны методы, необходимые для работы с функцией.

Метод “getLeftDomainBorder()” возвращает абсциссу самой левой (первой) точки

Метод “getRightDomainBorder()” аналогично возвращает значение самой правой (последней) точки

Метод “getFunctionValue(double x)” возвращает значение функции в точке  $x$ . Если точка лежит вне области определения функции, возвращается значение неопределенности, взятое из поля “NaN” класса “Double”, иначе проходим по абсциссам точек и сравниваем с входным  $x$ , когда мы нашли такую абсциссу, которая больше или равна входному  $x$ , проверяем предыдущую и текущую точку на равенство ее абсциссы с входным значением  $x$ , если нашли совпадение, возвращаем ординату точки, иначе используем линейную интерполяцию для нахождения ординаты.

## Ход выполнения задания номер 5

В классе “TabulatedFunction” были описаны методы, необходимые для работы с точками табулированной функции.

Метод “getPointsCount()” возвращает количество точек

Метод “FunctionPoint getPoint(int index)” возвращает копию точки, соответствующей переданному индексу. Для того чтобы передавать ссылку не на уже существующий объект класса “FunctionPoint”, что противоречило бы принципу инкапсуляции, нужно создать копию данной точки и возвращать ссылку на нее, для этого можно использовать конструктор, описанный в задании 2.

Метод “setPoint(int index, FunctionPoint point)” заменяет указанную точку на переданную. Аналогично предыдущему заданию, для того чтобы не нарушать принцип инкапсуляции, будем создавать копию переданной точки. В соответствии условиям задания, будем проверять соответствие точки области определения и интервалу соседних точек.

Метод “getPointX(int index)” возвращает значение абсциссы точки с указанным номером

Метод “setPointX(int index, double x)” аналогично методу “setPoint” изменяет значение абсциссы точки с указанным номером

Метод “getPointY(int index)” возвращает значение ординаты точки с указанным номером

Метод “setPointY(int index, double y)” изменяет значение ординаты точки с указанным номером

## Ход выполнения задания номер 6

В классе “TabulatedFunction” были описаны методы, необходимые для изменения количества точек табулированной функции.

Метод “deletePoint(int index)” удаляет заданную точку. Используем метод “arraycopy()” класса “System” чтобы перезаписать массив сам в себя, пропустив элемент с заданным индексом, после этого обнуляем последний элемент массива, чтобы он не дублировался с предпоследним и уменьшаем значение “pointsCount” на 1.

Метод “addPoint(FunctionPoint point)” добавляет новую точку. Будем проверять меньше ли фактическое количество точек, чем длина массива, если да, то нет надобности в создании нового массива, можно просто перезаписать его аналогично предыдущему заданию, добавив точку, если же количество точек равно длине массива, необходимо создать массив с длиной на 1 больше текущего. Остальная логика добавления точки схожа с той, что была в предыдущих методах.

## Ход выполнения задания номер 7

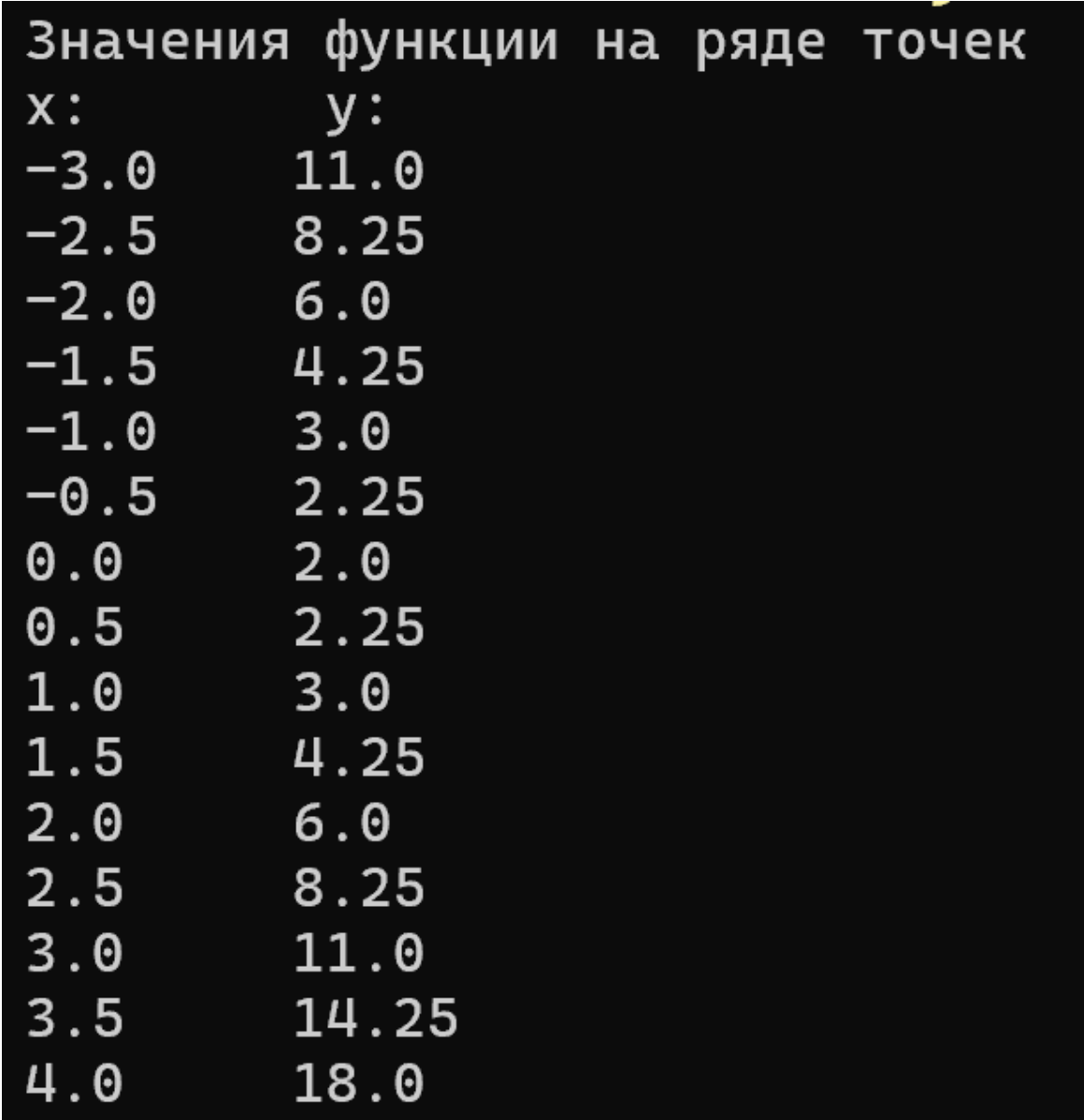
Согласно заданию, в пакете по умолчанию был создан класс “Main”, содержащий точку входа в программу. В классах “FunctionPoint” и “TabulatedFunction” пропишем “package functions;”, а в классе “Main” “import functions.FunctionPoint;

import functions.TabulatedFunction;”

Используем класс “Main” для проверки методов, написанных ранее.

Сначала создадим экземпляр класса “TabulatedFunction”, зададим область определения от -3 до 4 и количество точек = 15, функция:  $y = x^2 + 2$ .

Выведем в консоль значения функции на ряде точек (рис. 1)



x:	y:
-3.0	11.0
-2.5	8.25
-2.0	6.0
-1.5	4.25
-1.0	3.0
-0.5	2.25
0.0	2.0
0.5	2.25
1.0	3.0
1.5	4.25
2.0	6.0
2.5	8.25
3.0	11.0
3.5	14.25
4.0	18.0

Рис. 1

Попробуем изменить точки (рис. 2) следующими методами:

```
FunctionPoint point = new FunctionPoint(-10, 0);
```

```
tabFunct.setPoint(0, point);
```

```
tabFunct.setPoint(1, point);
```

```
tabFunct.setPointX(2, -2.3);
```

```
tabFunct.setPointY(2, 6.333);
```

```
tabFunct.setPointX(4, 100);
```

```
tabFunct.setPointY(4, 123);
```

Можно заметить, что точка с индексом 3 не была изменена, тк ее ордината не входит в интервал соседних точек, то же самое можно сказать и о точке с индексом 4, ее абсцисса не была изменена.



Изменяем точки	
х:	у:
-10.0	0.0
-2.5	8.25
-2.3	6.333
-1.5	4.25
-1.0	123.0
-0.5	2.25
0.0	2.0
0.5	2.25
1.0	3.0
1.5	4.25
2.0	6.0
2.5	8.25
3.0	11.0
3.5	14.25
4.0	18.0

Рис. 2

Попробуем удалить точку с индексом 0 (рис. 3). Видно, что удаление прошло успешно.

Удаляем точку	
х:	у:
-2.5	8.25
-2.3	6.333
-1.5	4.25
-1.0	123.0
-0.5	2.25
0.0	2.0
0.5	2.25
1.0	3.0
1.5	4.25
2.0	6.0
2.5	8.25
3.0	11.0
3.5	14.25
4.0	18.0

Рис. 3

Теперь попробуем добавить несколько точек (рис.4):

```
point = new FunctionPoint(3.12, 12.1);
```

```
tabFunct.addPoint(point);
```

```
point = new FunctionPoint(6, 38); // создаст новый массив, тк превысит  
размер изначального
```

```
tabFunct.addPoint(point);
```

Добавление точек прошло успешно.

Добавляем точки	
x:	y:
-2.5	8.25
-2.3	6.333
-1.5	4.25
-1.0	123.0
-0.5	2.25
0.0	2.0
0.5	2.25
1.0	3.0
1.5	4.25
2.0	6.0
2.5	8.25
3.0	11.0
3.12	12.1
3.5	14.25
4.0	18.0
6.0	38.0

Рис. 4

Проверим метод “getFunctionValue”, попробуем вернуть значение функции в точках: -2.5, 2.5, 5, 7 (рис. 5). Из рисунка видно, что значение функции в точке с абсциссой 5 было вычислено, а точка с абсциссой 7 выходит за границы области определения, поэтому было выведено значение неопределенности.

```
Вернем значение функции в точке x  
8.25  
8.25  
28.0  
NaN
```

Рис. 5