

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «САМАРСКИЙ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ имени
академика С.П. КОРОЛЁВА»

КАФЕДРА «ТЕХНИЧЕСКАЯ КИБЕРНЕТИКА»

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ

«Объектно-ориентированное программирование»

ЛАБОРАТОРНАЯ РАБОТА №2

Студент _____ Трофимов А. В.

Группа _____ 6301-030301D

Руководитель _____ Борисов Д. С.

Оценка _____

Задание 2

Создан класс `FunctionPoint`, инкапсулирующий понятие точки на графике функции. Класс обеспечивает хранение координат точки и предоставляет безопасные методы доступа к ним.



```
functions > J FunctionPoint.java > Language Support for Java(TM) by Red Hat > FunctionPoint > getY()
1 package functions;
2
3 public class FunctionPoint {
4     private double x;
5     private double y;
6
7     // Конструктор с заданными координатами
8     public FunctionPoint(double x, double y) {
9         this.x = x;
10        this.y = y;
11    }
12
13    // Конструктор копирования
14    public FunctionPoint(FunctionPoint point) {
15        this.x = point.x;
16        this.y = point.y;
17    }
18
19    // Конструктор по умолчанию (0, 0)
20    public FunctionPoint() {
21        this(x: 0.0, y: 0.0);
22    }
23
24    // Геттер для координаты x
25    public double getX() {
26        return x;
27    }
28
29    // Геттер для координаты y
30    public double getY() {
31        return y;
32    }
33
34    // Сеттер для координаты x
35    public void setX(double x) {
36        this.x = x;
37    }
38
39    // Сеттер для координаты y
40    public void setY(double y) {
41        this.y = y;
42    }
43 }
```

Задание 3

Создан класс `TabulatedFunction` для представления табулированной функции. Конструкторы обеспечивают различные способы инициализации функции с гарантией упорядоченности точек по координате x .

```
J Main.java    J TabulatedFunction.java X    J FunctionPoint.java
functions > J TabulatedFunction.java > Java > TabulatedFunction
1  package functions;
2
3  public class TabulatedFunction {
4      private FunctionPoint[] points;
5      private int pointsCount;
6
7      // Конструктор с количеством точек
8      public TabulatedFunction(double leftX, double rightX, int pointsCount) {
9          if (leftX >= rightX || pointsCount < 2) {
10             throw new IllegalArgumentException();
11          }
12
13          this.points = new FunctionPoint[pointsCount];
14          this.pointsCount = pointsCount;
15          double step = (rightX - leftX) / (pointsCount - 1);
16
17          for (int i = 0; i < pointsCount; i++) {
18              double x = leftX + i * step;
19              points[i] = new FunctionPoint(x, 0); // создает новую точку добавляет ее в массив points
20          }
21      }
22
23      // Конструктор с массивом значений
24      public TabulatedFunction(double leftX, double rightX, double[] values) { // задает точки когда есть у values массив y
25          if (leftX >= rightX || values.length < 2) {
26             throw new IllegalArgumentException();
27          }
28
29          this.points = new FunctionPoint[values.length];
30          this.pointsCount = values.length;
31          double step = (rightX - leftX) / (values.length - 1);
32
33          for (int i = 0; i < values.length; i++) {
34              double x = leftX + i * step;
35              points[i] = new FunctionPoint(x, values[i]);
36          }
37      }
38  }
```

Задание 4

Реализованы методы для определения границ области определения и вычисления значений функции с использованием линейной интерполяции.

```
39 // Методы области определения
40 public double getLeftDomainBorder() { // левая граница
41     return points[0].getX();
42 }
43
44 public double getRightDomainBorder() { // правая
45     return points[pointsCount - 1].getX();
46 }
47
48 public double getFunctionValue(double x) {
49     if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
50         return Double.NaN;
51     }
52
53     // Сначала проверяем, не совпадает ли x с одной из существующих точек
54     for (int i = 0; i < pointsCount; i++) {
55         if (Math.abs(points[i].getX() - x) < 1e-9) { // Используем машинный эпсилон для сравнения double
56             return points[i].getY(); // Возвращаем соответствующий y
57         }
58     }
59
60     // Если не совпадает ни с одной точкой, ищем интервал для интерполяции
61     for (int i = 0; i < pointsCount - 1; i++) { // поиск промежутка в котором лежит x
62         if (x >= points[i].getX() && x <= points[i + 1].getX()) {
63             double x1 = points[i].getX();
64             double y1 = points[i].getY();
65             double x2 = points[i + 1].getX();
66             double y2 = points[i + 1].getY();
67
68             return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
69         }
70     }
71
72     return Double.NaN;
73 }
```

Задание 5

Реализованы методы для безопасного доступа и модификации точек функции с сохранением инкапсуляции и упорядоченности.

```
67 // Методы работы с точками
68 public int getPointsCount() { // возвращает количество точек
69     return pointsCount;
70 }
71
72 public FunctionPoint getPoint(int index) { // возврат точки для соблюдения инкапсуляции
73     if (index < 0 || index >= pointsCount) {
74         throw new IndexOutOfBoundsException(); //индекс вне массива
75     }
76     return new FunctionPoint(points[index]); // возвращает копию точки на позиции индекса
77 }
78
79 public void setPoint(int index, FunctionPoint point) { // замена точки по индексу
80     if (index < 0 || index >= pointsCount) {
81         throw new IndexOutOfBoundsException();
82     }
83
84     if (point.getX() <= points[index - 1].getX() || point.getX() >= points[index + 1].getX()) {
85         throw new IllegalArgumentException(); // неправильный аргумент x
86     }
87
88     points[index] = new FunctionPoint(point); // ставится новая точка ху поинт
89 }
90
91 public double getPointX(int index) { // получает координату x по индексу
92     if (index < 0 || index >= pointsCount) {
93         throw new IndexOutOfBoundsException();
94     }
95     return points[index].getX(); // возвращает точку x
96 }
97
98 public void setPointX(int index, double x) { // заменяет координату x
99     if (index < 0 || index >= pointsCount) {
100         throw new IndexOutOfBoundsException();
101     }
102
103     if (x <= points[index - 1].getX() || x >= points[index + 1].getX()) { // проверка упорядоченности точки
104         throw new IllegalArgumentException();
105     }
106
107     points[index].setX(x); // на позицию индекса заменяет x у точки стоящей на позиции индекса
108 }
```

```
109
110 public double getPointY(int index) {
111     if (index < 0 || index >= pointsCount) {
112         throw new IndexOutOfBoundsException();
113     }
114     return points[index].getY(); // возврат y
115 }
116
117 public void setPointY(int index, double y) { //проверка индекса
118     if (index < 0 || index >= pointsCount) {
119         throw new IndexOutOfBoundsException();
120     }
121     points[index].setY(y); // замена y
122 }
```

Задание 6

Реализованы методы для динамического изменения количества точек с сохранением упорядоченности и эффективным использованием памяти.

```
124 // Методы изменения количества точек
125 public void deletePoint(int index) { // удаление точки
126     if (index < 0 || index >= pointsCount) {
127         throw new IndexOutOfBoundsException();
128     }
129     if (pointsCount < 3) {
130         throw new IllegalStateException();
131     }
132     // сдвигает массив влево
133     System.arraycopy(points, index + 1, points, index, pointsCount - index - 1); // 1(откуда копируем), 2(копирование точек с позиции индекс +1)
134     pointsCount--; // удаляем точку // 3(то куда копируем), 4(то откуда начинает вставлять)
135     points[pointsCount] = null; // зануляет точку на позиции pointsCount // 5(то сколько элементов нужно скопировать)
136 }
137
138 public void addPoint(FunctionPoint point) { // добавляет точку
139     // Поиск позиции для вставки
140     int insertIndex = 0;
141     while (insertIndex < pointsCount && points[insertIndex].getX() < point.getX()) { //ищет место куда вставить точку
142         insertIndex++;
143     }
144
145     // Проверка на дублирование x
146     if (insertIndex < pointsCount && Math.abs(points[insertIndex].getX() - point.getX()) < 1e-9) { //нельзя добавлять ту же точку выдаст ошибку
147         throw new IllegalArgumentException(); // конечное число в 10 может быть не конечным в 2 сс
148     }
149
150     // Проверка необходимости расширения массива
151     if (pointsCount == points.length) {
152         FunctionPoint[] newPoints = new FunctionPoint[points.length + 10];
153         System.arraycopy(points, 0, newPoints, 0, pointsCount);
154         points = newPoints;
155     }
156
157     // Сдвиг элементов вправо и вставка
158     System.arraycopy(points, insertIndex, points, insertIndex + 1, pointsCount - insertIndex);
159     points[insertIndex] = new FunctionPoint(point);
160     pointsCount++;
161 }
162 }
```

Задание 7

Реализованы методы для динамического изменения количества точек с сохранением упорядоченности и эффективным использованием памяти.

```
J Main.java X J TabulatedFunction.java J FunctionPoint.java
J Main.java > Java > Main > main(String[] args)
1  import functions.*;
2
3  public class Main {
4      Run main | Debug main | Run | Debug
      public static void main(String[] args) {
5          // Создаем табулированную функцию для  $x^2$  на интервале [0, 4]
6          double[] values = {0, 1, 4, 9, 16}; // создаем массив y
7          TabulatedFunction function = new TabulatedFunction(leftX: 0, rightX: 4, values);
8
9          System.out.println(x: "Исходная функция:");
10         printFunctionInfo(function); // выдаст все ху
11
12         // Проверяем значения функции в разных точках
13         System.out.println(x: "\nЗначения функции в разных точках:");
14         double[] testPoints = {-1, 0, 0.5, 1, 1.5, 2, 3, 4, 5};
15         for (double x : testPoints) { // по порядку из тест поинт
16             double y = function.getFunctionValue(x); // лин инт-я
17             System.out.printf(format: "f(%.1f) = %s\n", x, Double.isNaN(y) ? "NaN" : String.format(format: "%.2f", y));
18         }
19
20         // Изменяем точки
21         System.out.println(x: "\nПосле изменения точек:");
22         function.setPointY(index: 2, y: 5); // Меняем y в точке с индексом 2
23         function.setPoint(index: 3, new FunctionPoint(x: 3.7, y: 12.3)); // Меняем точку с индексом 3
24         printFunctionInfo(function);
25
26         // Добавляем точку
27         System.out.println(x: "\nПосле добавления точки (2.5, 6.3):");
28         function.addPoint(new FunctionPoint(x: 2.5, y: 6.3));
29         printFunctionInfo(function);
30
31         // Удаляем точку
32         System.out.println(x: "\nПосле удаления точки с индексом 1:");
33         function.deletePoint(index: 1);
34         printFunctionInfo(function);
35
36         // Проверяем значения после изменений
37         System.out.println(x: "\nЗначения после изменений:");
38         for (double x : testPoints) {
39             double y = function.getFunctionValue(x);
40             System.out.printf(format: "f(%.1f) = %s\n", x, Double.isNaN(y) ? "NaN" : String.format(format: "%.2f", y));
41         }
42     }
43     private static void printFunctionInfo(TabulatedFunction function) {
44         System.out.printf(format: "Область определения: [%.1f, %.1f]\n",
45             function.getLeftDomainBorder(), function.getRightDomainBorder());
46         System.out.println(x: "Точки функции:");
47         for (int i = 0; i < function.getPointsCount(); i++) {
48             FunctionPoint point = function.getPoint(i);
49             System.out.printf(format: " (%4.1f; %6.2f)\n", point.getX(), point.getY());
50         }
51     }
52 }
```

Исходная функция:

Область определения: $[0,0, 4,0]$

Точки функции:

```
( 0,0; 0,00)
( 1,0; 1,00)
( 2,0; 4,00)
( 3,0; 9,00)
( 4,0; 16,00)
```

Значения функции в разных точках:

$f(-1,0) = \text{NaN}$

$f(0,0) = 0,00$

$f(0,5) = 0,50$

$f(1,0) = 1,00$

$f(1,5) = 2,50$

$f(2,0) = 4,00$

$f(3,0) = 9,00$

$f(4,0) = 16,00$

$f(5,0) = \text{NaN}$

После изменения точек:

Область определения: $[0,0, 4,0]$

Точки функции:

```
( 0,0; 0,00)
( 1,0; 1,00)
( 2,0; 5,00)
( 3,7; 12,30)
( 4,0; 16,00)
```

После добавления точки (2.5, 6.3):

Область определения: $[0,0, 4,0]$

Точки функции:

```
( 0,0; 0,00)
( 1,0; 1,00)
( 2,0; 5,00)
( 2,5; 6,30)
( 3,7; 12,30)
( 4,0; 16,00)
```

После удаления точки с индексом 1:

Область определения: $[0,0, 4,0]$

Точки функции:

```
( 0,0; 0,00)
( 2,0; 5,00)
( 2,5; 6,30)
( 3,7; 12,30)
( 4,0; 16,00)
```

Значения после изменений:

$f(-1,0) = \text{NaN}$

$f(0,0) = 0,00$

$f(0,5) = 1,25$

$f(1,0) = 2,50$

$f(1,5) = 3,75$

$f(2,0) = 5,00$

$f(3,0) = 8,80$

$f(4,0) = 16,00$

$f(5,0) = \text{NaN}$

PS C:\Users\user\Desktop\ООП\Lab2>