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 _____

САМАРА 2025

Задание 2

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

Main.java | TabulatedFunction.java | FunctionPoint.java

functions > FunctionPoint.java > Language Support for Java(TM) by Red Hat > FunctionPoint > getY()

```java
package functions;

public class FunctionPoint {
    private double x;
    private double y;

    // Конструктор с заданными координатами
    public FunctionPoint(double x, double y) {
        this.x = x;
        this.y = y;
    }

    // Конструктор копирования
    public FunctionPoint(FunctionPoint point) {
        this.x = point.x;
        this.y = point.y;
    }

    // Конструктор по умолчанию (0, 0)
    public FunctionPoint() {
        this(x: 0.0, y: 0.0);
    }

    // Геттер для координаты x
    public double getX() {
        return x;
    }

    // Геттер для координаты y
    public double getY() {
        return y;
    }

    // Сеттер для координаты x
    public void setX(double x) {
        this.x = x;
    }

    // Сеттер для координаты y
    public void setY(double y) {
        this.y = y;
    }
}
```

Задание 3

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

Main.java | TabulatedFunction.java | FunctionPoint.java

functions > TabulatedFunction.java > Java > TabulatedFunction

```java
package functions;

public class TabulatedFunction {
    private FunctionPoint[] points;
    private int pointsCount;

    // Конструктор с количеством точек
    public TabulatedFunction(double leftX, double rightX, int pointsCount) {
        if (leftX >= rightX || pointsCount < 2) {
            throw new IllegalArgumentException();
        }

        this.points = new FunctionPoint[pointsCount];
        this.pointsCount = pointsCount;
        double step = (rightX - leftX) / (pointsCount - 1);

        for (int i = 0; i < pointsCount; i++) {
            double x = leftX + i * step;
            points[i] = new FunctionPoint(x, y: 0); // создает новую точку добавляет ее в массив поинтс
        }
    }

    // Конструктор с массивом значений
    public TabulatedFunction(double leftX, double rightX, double[] values) { // задает точки когда есть у vallues массив у
        if (leftX >= rightX || values.length < 2) {
            throw new IllegalArgumentException();
        }

        this.points = new FunctionPoint[values.length];
        this.pointsCount = values.length;
        double step = (rightX - leftX) / (values.length - 1);

        for (int i = 0; i < values.length; i++) {
            double x = leftX + i * step;
            points[i] = new FunctionPoint(x, values[i]);
        }
    }
```

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          for (int i = 0; i < pointsCount - 1; i++) { // поиск промежутка в котором лежит x
54              if (x >= points[i].getX() && x <= points[i + 1].getX()) {
55                  double x1 = points[i].getX();
56                  double y1 = points[i].getY();
57                  double x2 = points[i + 1].getX();
58                  double y2 = points[i + 1].getY();
59
60                  return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
61              }
62          }
63
64          return Double.NaN;
65      }
```

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 xy поинт
89      }
90
91      public double getPointX(int index) { // получает коорд 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 коорд 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 идекса заменяет x у точки стоящей на позиции индекс
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     }
```

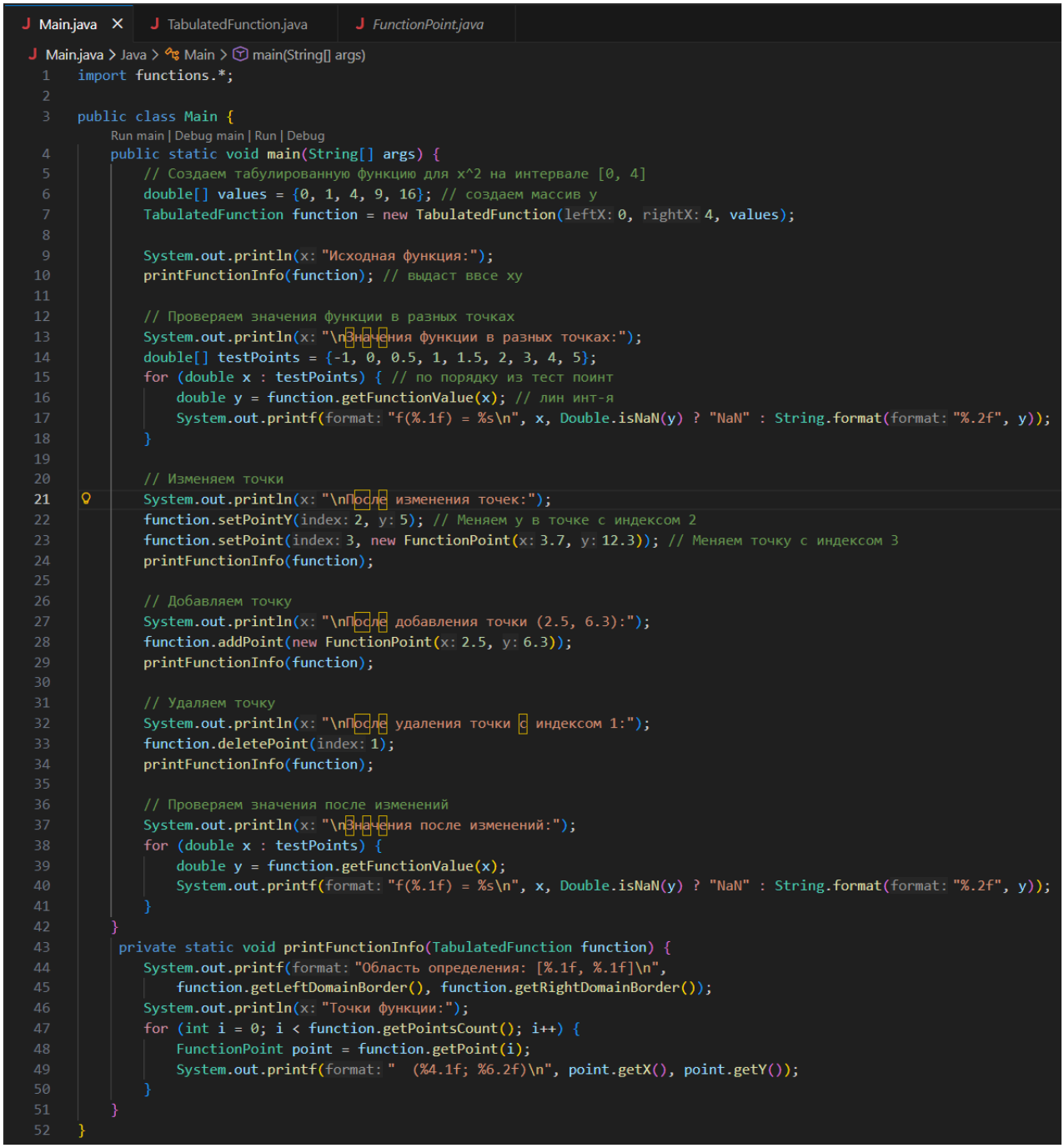
Задание 6

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

```java
    // Методы изменения количества точек
    public void deletePoint(int index) { // удаление точки
        if (index < 0 || index >= pointsCount) {
            throw new IndexOutOfBoundsException();
        }
        if (pointsCount < 3) {
            throw new IllegalStateException();
        }
        // сдвигает массив влево
        System.arraycopy(points, index + 1, points, index, pointsCount - index - 1);// 1(откуда копируем),2(копирование точек с позиции индекс +1)
        pointsCount--;   // удаляем точку                                            // 3(то куда копируе), 4(то откуда начинает вставлять)
        points[pointsCount] = null;  // зануляет точку на позиции поинтс каунт      // 5(то сколько эл ов нужно скопировать)
    }

    public void addPoint(FunctionPoint point) { // добавляет точку
        // Поиск позиции для вставки
        int insertIndex = 0;
        while (insertIndex < pointsCount && points[insertIndex].getX() < point.getX()) { //ищет место куда вставить точку
            insertIndex++;
        }

        // Проверка на дублирование x
        if (insertIndex < pointsCount && Math.abs(points[insertIndex].getX() - point.getX()) < 1e-9) {//нельзя добавлять ту же точку выдаст ошибку
            throw new IllegalArgumentException();     // конечное число в 10 может быть не конечным в 2 сс
        }

        // Проверка необходимости расширения массива
        if (pointsCount == points.length) {
            FunctionPoint[] newPoints = new FunctionPoint[points.length + 10];
            System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, pointsCount);
            points = newPoints;
        }

        // Сдвиг элементов вправо и вставка
        System.arraycopy(points, insertIndex, points, insertIndex + 1, pointsCount - insertIndex);
        points[insertIndex] = new FunctionPoint(point);
        pointsCount++;
    }
}
```

Задание 7

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



```java
import functions.*;

public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        // Создаем табулированную функцию для x^2 на интервале [0, 4]
        double[] values = {0, 1, 4, 9, 16}; // создаем массив y
        TabulatedFunction function = new TabulatedFunction(leftX: 0, rightX: 4, values);

        System.out.println(x: "Исходная функция:");
        printFunctionInfo(function); // выдаст ввсе xy

        // Проверяем значения функции в разных точках
        System.out.println(x: "\nЗначения функции в разных точках:");
        double[] testPoints = {-1, 0, 0.5, 1, 1.5, 2, 3, 4, 5};
        for (double x : testPoints) { // по порядку из тест поинт
            double y = function.getFunctionValue(x); // лин инт-я
            System.out.printf(format: "f(%.1f) = %s\n", x, Double.isNaN(y) ? "NaN" : String.format(format: "%.2f", y));
        }

        // Изменяем точки
        System.out.println(x: "\nПосле изменения точек:");
        function.setPointY(index: 2, y: 5); // Меняем y в точке с индексом 2
        function.setPoint(index: 3, new FunctionPoint(x: 3.7, y: 12.3)); // Меняем точку с индексом 3
        printFunctionInfo(function);

        // Добавляем точку
        System.out.println(x: "\nПосле добавления точки (2.5, 6.3):");
        function.addPoint(new FunctionPoint(x: 2.5, y: 6.3));
        printFunctionInfo(function);

        // Удаляем точку
        System.out.println(x: "\nПосле удаления точки с индексом 1:");
        function.deletePoint(index: 1);
        printFunctionInfo(function);

        // Проверяем значения после изменений
        System.out.println(x: "\nЗначения после изменений:");
        for (double x : testPoints) {
            double y = function.getFunctionValue(x);
            System.out.printf(format: "f(%.1f) = %s\n", x, Double.isNaN(y) ? "NaN" : String.format(format: "%.2f", y));
        }
    }
     private static void printFunctionInfo(TabulatedFunction function) {
        System.out.printf(format: "Область определения: [%.1f, %.1f]\n",
            function.getLeftDomainBorder(), function.getRightDomainBorder());
        System.out.println(x: "Точки функции:");
        for (int i = 0; i < function.getPointsCount(); i++) {
            FunctionPoint point = function.getPoint(i);
            System.out.printf(format: "  (%4.1f; %6.2f)\n", point.getX(), point.getY());
        }
    }
}
```