

# Лабораторная работа №2

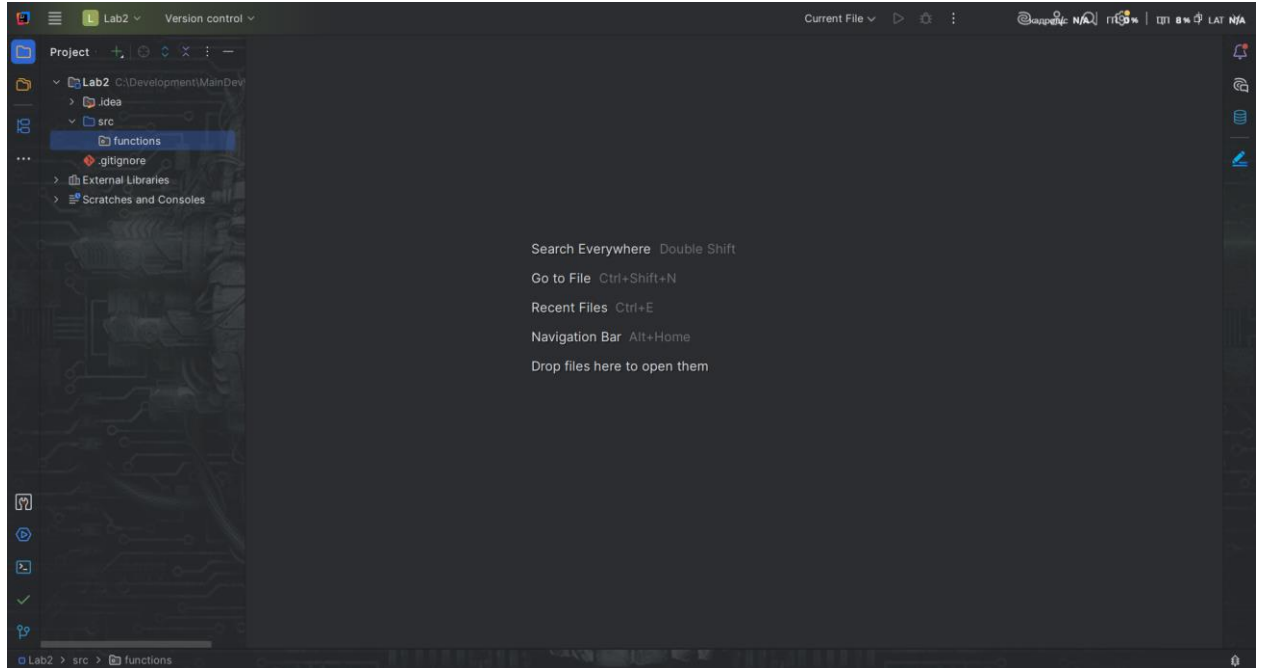
*Выполнил: Беляев Дмитрий Михайлович  
Студент 6203-010302D группы*

# Ход выполнения

Первым делом было прочитано ТЗ. Спасибо за внимание.

## Задание 1

Я смог создать пакет, результат предоставлен на скрине 1

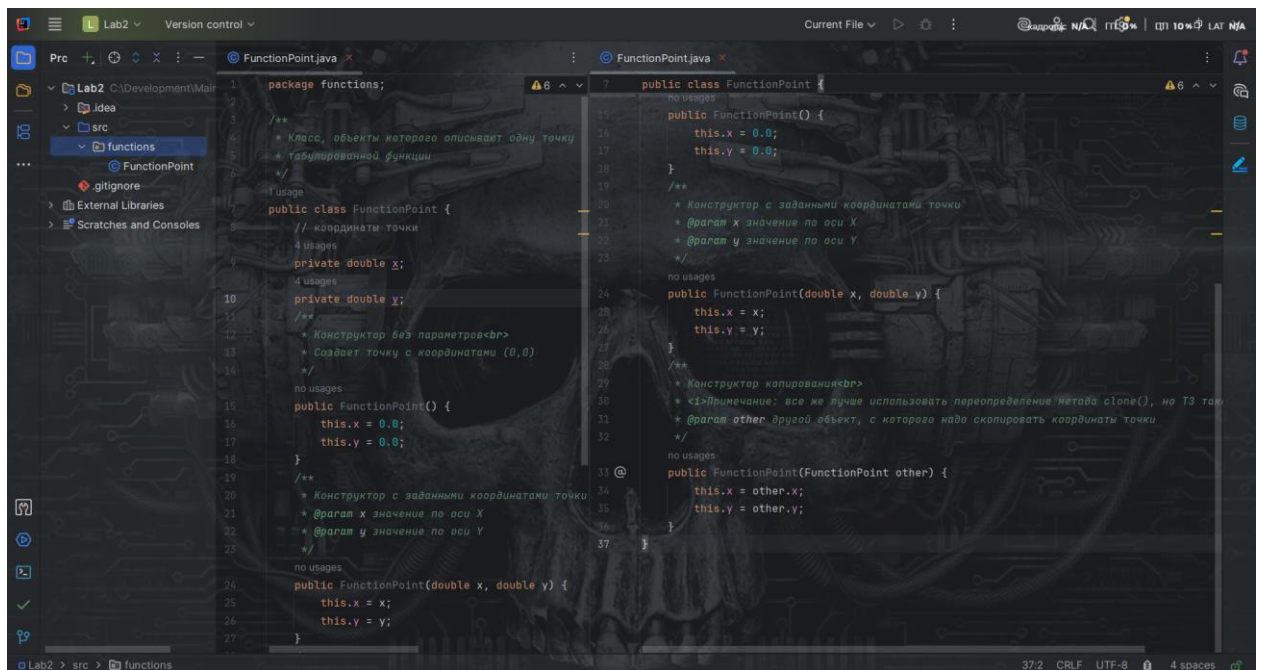


(скрин 1)

*Задание выполнено*

## Задание 2

Следующей моей задачей было создание класса FunctionPoint. Определил конструкторы, поля. Результат вы можете посмотреть по скрину 2.



(скрин 2)

Также от себя добавил геттеры и сеттеры(в будущем будет полезно). См скрин 3

```
public double getX() { return this.x; }

/**
 * Метод получения значения точки по оси Y
 * @return значение по оси Y
 */
no usages
public double getY() { return this.y; }

/**
 * Метод установки значения точки по оси X
 * @param x значение точки по оси X
 */
no usages
public void setX(double x) { this.x = x; }

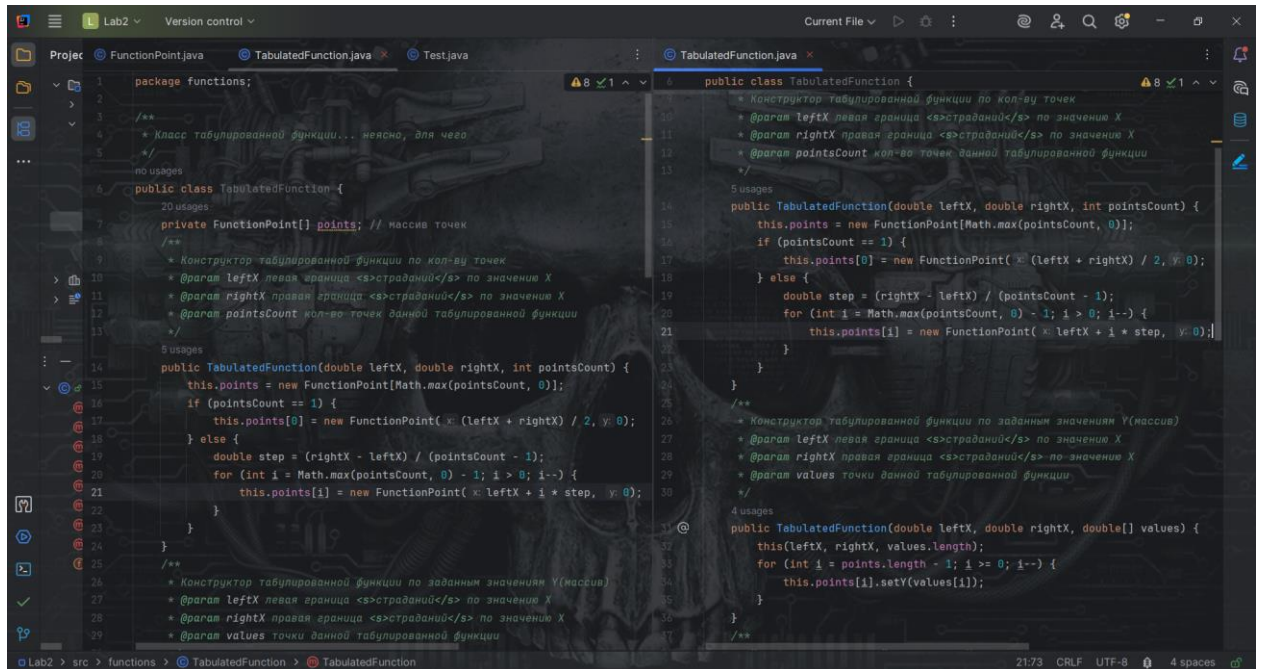
/**
 * Метод установки значения точки по оси Y
 * @param y значение точки по оси Y
 */
no usages
public void setY(double y) { this.y = y; }
```

(скрин 3)

Задание выполнено

### Задание 3

Были созданы конструкторы, также расставлены комментарии... см скрин 4-5  
(допустил ошибку, на задании 5 исправил, должно быть условие " $i > 0$ " вместо " $i \geq 0$ ")



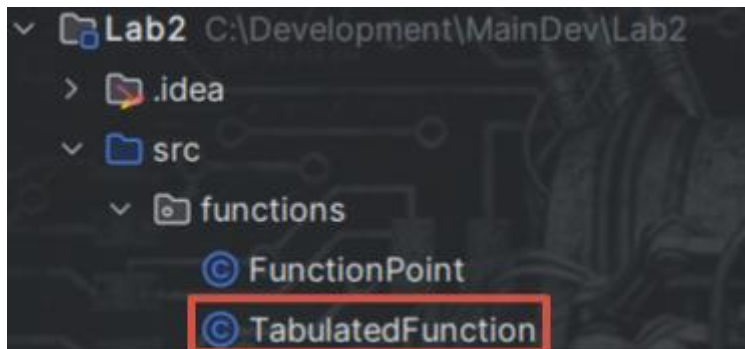
```
package functions;

/**
 * Класс табулированной функции... неясно, для чего
 */
public class TabulatedFunction {
    private FunctionPoint[] points; // массив точек

    /**
     * Конструктор табулированной функции по кол-ву точек
     * @param leftX левая граница <с>традиций</с> по значению X
     * @param rightX правая граница <с>традиций</с> по значению X
     * @param pointsCount кол-во точек данной табулированной функции
     */
    public TabulatedFunction(double leftX, double rightX, int pointsCount) {
        this.points = new FunctionPoint[Math.max(pointsCount, 0)];
        if (pointsCount == 1) {
            this.points[0] = new FunctionPoint((leftX + rightX) / 2, 0);
        } else {
            double step = (rightX - leftX) / (pointsCount - 1);
            for (int i = Math.max(pointsCount, 0) - 1; i > 0; i--) {
                this.points[i] = new FunctionPoint((leftX + i * step), 0);
            }
        }
    }

    /**
     * Конструктор табулированной функции по заданным значениям Y (массив)
     * @param leftX левая граница <с>традиций</с> по значению X
     * @param rightX правая граница <с>традиций</с> по значению X
     * @param values точки данной табулированной функции
     */
    public TabulatedFunction(double leftX, double rightX, double[] values) {
        this(leftX, rightX, values.length);
        for (int i = points.length - 1; i >= 0; i--) {
            this.points[i].setY(values[i]);
        }
    }
}
```

(скрин 4)



(скрин 5)

Задание выполнено

### Задание 4

В данный класс я добавил еще 3 метода, на которые вы можете взглянуть по скринам 6-7

```

/**
 * Получение значения в данной точке
 * @param x точка по X
 * @return интерполированное линейно значение в данной точке(Y)
 */
no usages
public double getFunctionValue(double x) {
    if (x < this.getLeftDomainBorder() || this.getRightDomainBorder() < x) return Double.NaN;
    int index;
    for (index = points.length - 2; index >= 0; index--) { // нет смысла проверять правую точку, скипаем автоматом
        if (this.points[index].getX() < x) break; // простите Александр Викторович, но вы это не должны видеть.
        if (this.points[index].getX() == x) return this.points[index].getY(); // нет смысла интерполировать
    } // здесь используется функция (y1-y0)/(x1-x0)*(x-x0) + y0
    return (this.points[index+1].getY() - this.points[index].getY()) /
        (this.points[index+1].getX() - this.points[index].getX()) *
        (x - this.points[index].getX()) + this.points[index].getY();
}

```

(скрин 6)

```

/**
 * Метод получения левой границы по X
 * @return левая граница X
 */
1 usage
public double getLeftDomainBorder() {
    return (this.points.length > 0) ? (this.points[0].getX()) : (Double.NaN);
}

/**
 * Метод получения правой границы по X
 * @return правая граница X
 */
1 usage
public double getRightDomainBorder() {
    return (this.points.length > 0) ? (this.points[this.points.length - 1].getX()) : (Double.NaN);
}

```

(скрин 7)

Задание выполнено

## Задание 5

Добавил согласно ТЗ методы(и от себя проверки, все равно они будут нужны). См скрины 8-10



```

/**
 * Метод получения кол-ва точек в данном классе(объекте)
 * @return кол-во точек в данном классе(объекте)
 */
no usages
public int getPointsCount() { return this.points.length; }

/**
 * Метод получения точки через индекс
 * @param index индекс получаемой точки
 * @return копия точки
 */
no usages
public FunctionPoint getPoint(int index) {
    return (index < 0 || index >= this.points.length) ?
        (null) : (new FunctionPoint(this.points[index]));
}

/**
 * Метод установки точки по заданному индексу
 * @param index индекс точки
 * @param point сама точка. Будет сохранена ее копия
 */
1 usage
public void setPoint(int index, FunctionPoint point) {
    if (index < 0 || index >= this.points.length) return;
    double leftX = (index == 0) ? (Double.NEGATIVE_INFINITY) : (this.points[index-1].getX());
    double rightX = (index == this.points.length - 1) ? (Double.POSITIVE_INFINITY) : (this.points[index+1].getX());
    if (leftX < point.getX() && point.getX() < rightX) this.points[index] = new FunctionPoint(point);
}

```

(скрин 8)

```

/**
 * Метод получения X координаты точки на index позиции
 * @param index индекс получаемой точки
 * @return значение точки по координате X
 */
no usages
public double getPointX(int index) {
    return (index < 0 || index >= this.points.length) ? (Double.NaN) : (this.points[index].getX());
}

/**
 * Метод изменения у точки X координаты
 * @param index индекс, чьей точки мы меняем X
 * @param x значение X координаты
 */
no usages
public void setPointX(int index, double x) {
    if (index >= 0 && index < this.points.length)
        this.setPoint(index, new FunctionPoint(x, this.points[index].getY()));
}

/**
 * Метод получения Y координаты точки на index позиции
 * @param index индекс получаемой точки
 * @return значение точки по координате Y
 */
no usages
public double getPointY(int index) {
    return (index < 0 || index >= this.points.length) ? (Double.NaN) : (this.points[index].getY());
}

```

(скрин 9)

```

/**
 * Метод изменения у точки Y координаты
 * @param index индекс, чьей точки мы меняем Y
 * @param y значение Y координаты
 */
no usages
public void setPointY(int index, double y) {
    if (index >= 0 && index < this.points.length) this.points[index].setY(y);
}

```

(скрин 10)

*Задание выполнено*

## Задание 6

Задание крайне сложное в плане оптимизации. Поэтому придется внести правки.

Правка №1 – добавим приватное поле, отображающий кол-во точек(см скрин 11)

Правка №2 – добавим в конструкторе установку кол-ва точек в массиве(см скрин 12)

Серия правок №3-8 – см скрины 13-18

```

public class TabulatedFunction {
    36 usages
    private FunctionPoint[] points; // массив точек
    no usages
    private int amountOfPoints; // реальное кол-во точек в данном массиве
    /**
     * Конструктор табулированной функции по кол-ву точек

```

(скрин 11)

```

public TabulatedFunction(double leftX, double rightX, int pointsCount) {
    this.points = new FunctionPoint[Math.max(pointsCount, 0)];
    this.amountOfPoints = pointsCount;
    if (pointsCount == 1) {
        this.points[0] = new FunctionPoint(x: (leftX + rightX) / 2, y: 0);
    } else {
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = Math.max(pointsCount, 0) - 1; i > 0; i--) {
            this.points[i] = new FunctionPoint(x: leftX + i * step, y: 0);
        }
    }
}

```

(скрин 12)

```

public TabulatedFunction(double leftX, double rightX, double[] values) {
    this(leftX, rightX, values.length);
    for (int i = this.amountOfPoints - 1; i >= 0; i--) {
        this.points[i].setY(values[i]);
    }
}

```

(скрин 13)

```

/**
 * Метод получения левой границы по X
 * @return левая граница X
 */
1 usage
public double getLeftDomainBorder() {
    return this.amountOfPoints > 0 ? (this.points[0].getX()) : (Double.NaN);
}

/**
 * Метод получения правой границы по X
 * @return правая граница X
 */
1 usage
public double getRightDomainBorder() {
    return (this.amountOfPoints > 0) ? (this.points[this.amountOfPoints - 1].getX()) : (Double.NaN);
}

```

(скрин 14)



```

/**
 * Получение значения в данной точке
 * @param x точка по X
 * @return интерполированное линейно значение в данной точке(Y)
 */
no usages
public double getFunctionValue(double x) {
    if (x < this.getLeftDomainBorder() || this.getRightDomainBorder() < x) return Double.NaN;
    int index;
    for (index = this.amountOfPoints - 2; index >= 0; index--) { // нет смысла проверять правую точку
        if (this.points[index].getX() < x) break; // простите Александр Викторович, но вы это не долж
        if (this.points[index].getX() == x) return this.points[index].getY(); // нет смысла интерполи
    } // здесь используется функция (y1-y0)/(x1-x0)*(x-x0) + y0
    return (this.points[index+1].getY() - this.points[index].getY()) /
        (this.points[index+1].getX() - this.points[index].getX()) *
        (x - this.points[index].getX()) + this.points[index].getY();
}

```

(скрин 15)

```

/**
 * Метод получения кол-ва точек в данном классе(объекте)
 * @return кол-во точек в данном классе(объекте)
 */
no usages
public int getPointsCount() { return this.amountOfPoints; }

/**
 * Метод получения точки через индекс
 * @param index индекс получаемой точки
 * @return копия точки
 */
no usages
public FunctionPoint getPoint(int index) {
    return (index < 0 || index >= this.amountOfPoints) ?
        (null) : (new FunctionPoint(this.points[index]));
}

```

(скрин 16)

```

public void setPoint(int index, FunctionPoint point) {
    if (index < 0 || index >= this.amountOfPoints) return;
    double leftX = (index == 0) ? (Double.NEGATIVE_INFINITY) : (this.points[index-1].getX());
    double rightX = (index == this.amountOfPoints - 1) ? (Double.POSITIVE_INFINITY) : (this.points[index+1].getX());
    if (leftX < point.getX() && point.getX() < rightX) this.points[index] = new FunctionPoint(point);
}

/**
 * Метод получения X координаты точки на index позиции
 * @param index индекс получаемой точки
 * @return значение точки по координате X
 */
no usages
public double getPointX(int index) {
    return (index < 0 || index >= this.amountOfPoints) ? (Double.NaN) : (this.points[index].getX());
}

/**
 * Метод изменения у точки X координаты
 * @param index индекс, чьей точки мы меняем X
 * @param x значение X координаты
 */
no usages
public void setPointX(int index, double x) {
    if (index >= 0 && index < this.amountOfPoints)
        this.setPoint(index, new FunctionPoint(x, this.points[index].getY()));
}

```

(скрин 17)

```

/**
 * Метод получения Y координаты точки на index позиции
 * @param index индекс получаемой точки
 * @return значение точки по координате Y
 */
no usages
public double getPointY(int index) {
    return (index < 0 || index >= this.amountOfPoints) ? (Double.NaN) : (this.points[index].getY());
}

/**
 * Метод изменения у точки Y координаты
 * @param index индекс, чьей точки мы меняем Y
 * @param y значение Y координаты
 */
no usages
public void setPointY(int index, double y) {
    if (index >= 0 && index < this.amountOfPoints) this.points[index].setY(y);
}

```

(скрин 18)

Правки внесены. Осталось добавить метод удаления точек(см скрин 19 реализация)

```

/**
 * Метод удаления точки по заданному индексу
 * @param index индекс удаляемой точки
 */
no usages
public void deletePoint(int index) {
    if (index < 0 || index >= this.amountOfPoints) return;
    for (; index < this.amountOfPoints - 1; index++) {
        this.points[index] = this.points[index + 1];
    }
    this.amountOfPoints--;
    this.points[index + 1] = null; /* стираем ссылку на последний элемент, т.к. если удалим последний элемент
    то из памяти он не исчезнет(ссылка останется на неиспользуемой части массива) */
}

```

(скрин 19)

Самое сложное – добавление точек. Реализация предоставлена на скрине 20

```

/**
 * Метод добавления точки в список
 * @param point сама собственно точка
 */
no usages
public void addPoint(FunctionPoint point) {
    double x = point.getX();
    int index = 0;
    for (; index < this.amountOfPoints; index++) {
        if (this.points[index].getX() == x) return; // если X совпадает - не добавляем точку.(идет она нафиг)
        if (this.points[index].getX() > x) break;
    }
    if (this.points.length == this.amountOfPoints) {
        FunctionPoint[] newArray = new FunctionPoint[this.points.length * 2 + 1];
        System.arraycopy(this.points, srcPos: 0, newArray, destPos: 0, index);
        newArray[index] = new FunctionPoint(point);
        System.arraycopy(this.points, index, newArray, destPos: index + 1, length: this.amountOfPoints - index - 2);
        this.points = newArray; // по идее массив удалится, т.к. нету на него ссылок
    } else {
        for (int i = this.amountOfPoints; i > index; i--) {
            this.points[i] = this.points[i - 1];
        }
        this.points[index] = new FunctionPoint(point);
    }
    this.amountOfPoints++;
}

```

(скрин 20)

*Задание выполнено*

## Задание 7

Сделаем Main класс, в нем расположим точку входа в программу, а также расположим тестирующие алгоритмы(просто проверка на работоспособность). См реализацию на скрине 23. В процессе выявил ошибки, исправил(см скрины 21-22). Ответ от программы предоставлен в таблице 1



```

public TabulatedFunction(double leftX, double rightX, int pointsCount) {
    this.points = new FunctionPoint[Math.max(pointsCount, 0)];
    this.amountOfPoints = pointsCount;
    if (pointsCount == 1) {
        this.points[0] = new FunctionPoint( x: (leftX + rightX) / 2, y: 0);
    } else {
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = Math.max(pointsCount, 0) - 1; i >= 0; i--) {
            this.points[i] = new FunctionPoint( x: leftX + i * step, y: 0);
        }
    }
}
}

```

(скрин 21)

```

/**
 * Метод удаления точки по заданному индексу
 * @param index индекс удаляемой точки
 */
1 usage
public void deletePoint(int index) {
    if (index < 0 || index >= this.amountOfPoints) return;
    for (; index < this.amountOfPoints - 2; index++) {
        this.points[index] = this.points[index + 1];
    }
    this.amountOfPoints--;
    this.points[index + 1] = null; /* стираем ссылку на последний элемент, т.к. если удалим последний элемент,
    то из памяти он не исчезнет(ссылка останется на неиспользуемой части массива) */
}

```

(скрин 22)

```

// Main.java
import functions.FunctionPoint;
import functions.TabulatedFunction;

public class Main {
    public static final double deltaX = 0.1;
    public static final double initialX = -1;
    public static final double endX = 3;

    public static void main(String[] args) {
        double[] arr = {1,4,9,16,25,36,49,64,81,100,121};
        TabulatedFunction tf = new TabulatedFunction( leftX: 1, rightX: 11, arr);

        System.out.println("Тестируем getFunctionValue...");
        display(tf);

        System.out.println();
        System.out.println("Смотрим, что при замене будет...");
        tf.setPoint( index: 0, new FunctionPoint( x: -1, y: 1));
        display(tf);

        System.out.println();
        System.out.println("Возвращаем точку обратно...");
        tf.setPoint( index: 0, new FunctionPoint( x: 1, y: 1));
        display(tf);

        System.out.println();
        System.out.println("Изменим одну точку...");
        tf.setPointX( index: 0, x: -5); // <----
        display(tf);
    }

    public static void display(TabulatedFunction tf) {
        double currentX = initialX;
        for (; currentX <= endX; currentX += deltaX) {
            System.out.println(currentX + " -> " + tf.getFunctionValue(currentX));
        }
    }
}

// TabulatedFunction.java
public TabulatedFunction(double leftX, double rightX, int pointsCount) {
    this.points = new FunctionPoint[Math.max(pointsCount, 0)];
    this.amountOfPoints = pointsCount;
    if (pointsCount == 1) {
        this.points[0] = new FunctionPoint( x: (leftX + rightX) / 2, y: 0);
    } else {
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = Math.max(pointsCount, 0) - 1; i >= 0; i--) {
            this.points[i] = new FunctionPoint( x: leftX + i * step, y: 0);
        }
    }
}

public void addPoint(int index, FunctionPoint point) {
    if (index < 0 || index >= this.amountOfPoints) return;
    this.points[index] = point;
    this.amountOfPoints++;
}

public void deletePoint(int index) {
    if (index < 0 || index >= this.amountOfPoints) return;
    for (; index < this.amountOfPoints - 2; index++) {
        this.points[index] = this.points[index + 1];
    }
    this.amountOfPoints--;
    this.points[index + 1] = null; /* стираем ссылку на последний элемент, т.к. если удалим последний элемент,
    то из памяти он не исчезнет(ссылка останется на неиспользуемой части массива) */
}

public double getFunctionValue(double x) {
    double currentX = initialX;
    double y = 0;
    for (; currentX <= endX; currentX += deltaX) {
        y = currentX * currentX;
        currentX += deltaX;
    }
    return y;
}

```

(скрин 23)



Номер	Детали
1	<p>Тестируем getFunctionValue...</p> <p>-1.0 -&gt; NaN</p> <p>-0.9 -&gt; NaN</p> <p>-0.8 -&gt; NaN</p> <p>-0.7000000000000001 -&gt; NaN</p> <p>-0.6000000000000001 -&gt; NaN</p> <p>-0.5000000000000001 -&gt; NaN</p> <p>-0.40000000000000013 -&gt; NaN</p> <p>-0.30000000000000016 -&gt; NaN</p> <p>-0.20000000000000015 -&gt; NaN</p> <p>-0.10000000000000014 -&gt; NaN</p> <p>-1.3877787807814457E-16 -&gt; NaN</p> <p>0.09999999999999987 -&gt; NaN</p> <p>0.19999999999999987 -&gt; NaN</p> <p>0.2999999999999999 -&gt; NaN</p> <p>0.3999999999999999 -&gt; NaN</p> <p>0.4999999999999999 -&gt; NaN</p> <p>0.5999999999999999 -&gt; NaN</p> <p>0.6999999999999998 -&gt; NaN</p> <p>0.7999999999999998 -&gt; NaN</p> <p>0.8999999999999998 -&gt; NaN</p> <p>0.9999999999999998 -&gt; NaN</p> <p>1.0999999999999999 -&gt; 1.2999999999999996</p> <p>1.2 -&gt; 1.5999999999999999</p> <p>1.3 -&gt; 1.9000000000000001</p> <p>1.4000000000000001 -&gt; 2.2</p> <p>1.5000000000000002 -&gt; 2.5000000000000001</p> <p>1.6000000000000003 -&gt; 2.8000000000000007</p> <p>1.7000000000000004 -&gt; 3.10000000000000014</p> <p>1.8000000000000005 -&gt; 3.40000000000000012</p> <p>1.9000000000000006 -&gt; 3.7000000000000002</p> <p>2.0000000000000004 -&gt; 4.0000000000000002</p> <p>2.1000000000000005 -&gt; 4.5000000000000003</p> <p>2.2000000000000006 -&gt; 5.00000000000000036</p> <p>2.3000000000000007 -&gt; 5.50000000000000036</p> <p>2.4000000000000001 -&gt; 6.00000000000000036</p> <p>2.5000000000000001 -&gt; 6.5000000000000004</p> <p>2.6000000000000001 -&gt; 7.0000000000000005</p> <p>2.7000000000000001 -&gt; 7.5000000000000005</p> <p>2.8000000000000001 -&gt; 8.0000000000000005</p> <p>2.90000000000000012 -&gt; 8.5000000000000007</p>
2	<p>Смотрим, что при замене будет...</p> <p>-1.0 -&gt; 1.0</p> <p>-0.9 -&gt; 1.1</p> <p>-0.8 -&gt; 1.2</p> <p>-0.7000000000000001 -&gt; 1.2999999999999998</p> <p>-0.6000000000000001 -&gt; 1.4</p> <p>-0.5000000000000001 -&gt; 1.5</p>

	-0.400000000000000013 -> 1.5999999999999999 -0.300000000000000016 -> 1.6999999999999997 -0.200000000000000015 -> 1.7999999999999998 -0.100000000000000014 -> 1.9 -1.3877787807814457E-16 -> 2.0 0.09999999999999987 -> 2.0999999999999996 0.19999999999999987 -> 2.2 0.2999999999999999 -> 2.3 0.3999999999999999 -> 2.4 0.4999999999999999 -> 2.5 0.5999999999999999 -> 2.5999999999999996 0.6999999999999998 -> 2.6999999999999997 0.7999999999999998 -> 2.8 0.8999999999999998 -> 2.9 0.9999999999999998 -> 3.0 1.0999999999999999 -> 3.0999999999999996 1.2 -> 3.2 1.3 -> 3.3 1.40000000000000001 -> 3.40000000000000004 1.50000000000000002 -> 3.5 1.60000000000000003 -> 3.60000000000000005 1.70000000000000004 -> 3.7 1.80000000000000005 -> 3.80000000000000007 1.90000000000000006 -> 3.90000000000000004 2.00000000000000004 -> 4.00000000000000002 2.10000000000000005 -> 4.50000000000000003 2.20000000000000006 -> 5.000000000000000036 2.30000000000000007 -> 5.500000000000000036 2.40000000000000001 -> 6.000000000000000036 2.50000000000000001 -> 6.50000000000000004 2.60000000000000001 -> 7.00000000000000005 2.70000000000000001 -> 7.50000000000000005 2.80000000000000001 -> 8.00000000000000005 2.900000000000000012 -> 8.50000000000000007
3	Возвращаем точку обратно... -1.0 -> NaN -0.9 -> NaN -0.8 -> NaN -0.70000000000000001 -> NaN -0.60000000000000001 -> NaN -0.50000000000000001 -> NaN -0.400000000000000013 -> NaN -0.300000000000000016 -> NaN -0.200000000000000015 -> NaN -0.100000000000000014 -> NaN -1.3877787807814457E-16 -> NaN 0.09999999999999987 -> NaN 0.19999999999999987 -> NaN 0.2999999999999999 -> NaN

	0.399999999999999999 -> NaN 0.499999999999999999 -> NaN 0.599999999999999999 -> NaN 0.699999999999999998 -> NaN 0.799999999999999998 -> NaN 0.899999999999999998 -> NaN 0.999999999999999998 -> NaN 1.099999999999999999 -> 1.299999999999999996 1.2 -> 1.599999999999999999 1.3 -> 1.900000000000000001 1.400000000000000001 -> 2.2 1.500000000000000002 -> 2.500000000000000001 1.600000000000000003 -> 2.800000000000000007 1.700000000000000004 -> 3.100000000000000014 1.800000000000000005 -> 3.400000000000000012 1.900000000000000006 -> 3.700000000000000002 2.000000000000000004 -> 4.000000000000000002 2.100000000000000005 -> 4.500000000000000003 2.200000000000000006 -> 5.000000000000000036 2.300000000000000007 -> 5.500000000000000036 2.400000000000000001 -> 6.000000000000000036 2.500000000000000001 -> 6.500000000000000004 2.600000000000000001 -> 7.000000000000000005 2.700000000000000001 -> 7.500000000000000005 2.800000000000000001 -> 8.000000000000000005 2.900000000000000012 -> 8.500000000000000007
4	Изменим одну точку... -1.0 -> 2.7142857142857144 -0.9 -> 2.757142857142857 -0.8 -> 2.8 -0.700000000000000001 -> 2.8428571428571425 -0.600000000000000001 -> 2.8857142857142857 -0.500000000000000001 -> 2.9285714285714284 -0.400000000000000013 -> 2.971428571428571 -0.300000000000000016 -> 3.0142857142857142 -0.200000000000000015 -> 3.057142857142857 -0.100000000000000014 -> 3.0999999999999996 -1.3877787807814457E-16 -> 3.142857142857143 0.099999999999999987 -> 3.1857142857142855 0.199999999999999987 -> 3.2285714285714286 0.299999999999999999 -> 3.2714285714285714 0.399999999999999999 -> 3.3142857142857145 0.499999999999999999 -> 3.357142857142857 0.599999999999999999 -> 3.4 0.699999999999999998 -> 3.4428571428571426 0.799999999999999998 -> 3.4857142857142853 0.899999999999999998 -> 3.528571428571428 0.999999999999999998 -> 3.571428571428571 1.099999999999999999 -> 3.614285714285714

	1.2 -> 3.657142857142857 1.3 -> 3.699999999999997 1.4000000000000001 -> 3.742857142857143 1.5000000000000002 -> 3.7857142857142856 1.6000000000000003 -> 3.8285714285714287 1.7000000000000004 -> 3.8714285714285714 1.8000000000000005 -> 3.9142857142857146 1.9000000000000006 -> 3.9571428571428573 2.0000000000000004 -> 4.0000000000000002 2.1000000000000005 -> 4.5000000000000003 2.2000000000000006 -> 5.00000000000000036 2.3000000000000007 -> 5.50000000000000036 2.4000000000000001 -> 6.00000000000000036 2.5000000000000001 -> 6.5000000000000004 2.6000000000000001 -> 7.0000000000000005 2.7000000000000001 -> 7.5000000000000005 2.8000000000000001 -> 8.0000000000000005 2.90000000000000012 -> 8.5000000000000007
5	Вернем обратно... -1.0 -> NaN -0.9 -> NaN -0.8 -> NaN -0.7000000000000001 -> NaN -0.6000000000000001 -> NaN -0.5000000000000001 -> NaN -0.40000000000000013 -> NaN -0.30000000000000016 -> NaN -0.20000000000000015 -> NaN -0.10000000000000014 -> NaN -1.3877787807814457E-16 -> NaN 0.09999999999999987 -> NaN 0.19999999999999987 -> NaN 0.2999999999999999 -> NaN 0.3999999999999999 -> NaN 0.4999999999999999 -> NaN 0.5999999999999999 -> NaN 0.6999999999999998 -> NaN 0.7999999999999998 -> NaN 0.8999999999999998 -> NaN 0.9999999999999998 -> NaN 1.0999999999999999 -> 1.2999999999999996 1.2 -> 1.5999999999999999 1.3 -> 1.9000000000000001 1.4000000000000001 -> 2.2 1.5000000000000002 -> 2.5000000000000001 1.6000000000000003 -> 2.8000000000000007 1.7000000000000004 -> 3.10000000000000014 1.8000000000000005 -> 3.40000000000000012 1.9000000000000006 -> 3.7000000000000002



	2.0000000000000004 -> 4.0000000000000002 2.1000000000000005 -> 4.5000000000000003 2.2000000000000006 -> 5.00000000000000036 2.3000000000000007 -> 5.50000000000000036 2.4000000000000001 -> 6.00000000000000036 2.5000000000000001 -> 6.5000000000000004 2.6000000000000001 -> 7.0000000000000005 2.7000000000000001 -> 7.5000000000000005 2.8000000000000001 -> 8.0000000000000005 2.90000000000000012 -> 8.5000000000000007
6	Удалим точку... -1.0 -> NaN -0.9 -> NaN -0.8 -> NaN -0.7000000000000001 -> NaN -0.6000000000000001 -> NaN -0.5000000000000001 -> NaN -0.40000000000000013 -> NaN -0.30000000000000016 -> NaN -0.20000000000000015 -> NaN -0.10000000000000014 -> NaN -1.3877787807814457E-16 -> NaN 0.09999999999999987 -> NaN 0.19999999999999987 -> NaN 0.2999999999999999 -> NaN 0.3999999999999999 -> NaN 0.4999999999999999 -> NaN 0.5999999999999999 -> NaN 0.6999999999999998 -> NaN 0.7999999999999998 -> NaN 0.8999999999999998 -> NaN 0.9999999999999998 -> NaN 1.0999999999999999 -> NaN 1.2 -> NaN 1.3 -> NaN 1.4000000000000001 -> NaN 1.5000000000000002 -> NaN 1.6000000000000003 -> NaN 1.7000000000000004 -> NaN 1.8000000000000005 -> NaN 1.9000000000000006 -> NaN 2.0000000000000004 -> 4.0000000000000002 2.1000000000000005 -> 4.5000000000000003 2.2000000000000006 -> 5.00000000000000036 2.3000000000000007 -> 5.50000000000000036 2.4000000000000001 -> 6.00000000000000036 2.5000000000000001 -> 6.5000000000000004 2.6000000000000001 -> 7.0000000000000005 2.7000000000000001 -> 7.5000000000000005

	2.8000000000000001 -> 8.000000000000005 2.9000000000000012 -> 8.500000000000007
7	Вернем обратно точку... -1.0 -> NaN -0.9 -> NaN -0.8 -> NaN -0.7000000000000001 -> NaN -0.6000000000000001 -> NaN -0.5000000000000001 -> NaN -0.4000000000000013 -> NaN -0.3000000000000016 -> NaN -0.2000000000000015 -> NaN -0.1000000000000014 -> NaN -1.3877787807814457E-16 -> NaN 0.0999999999999987 -> NaN 0.1999999999999987 -> NaN 0.299999999999999 -> NaN 0.399999999999999 -> NaN 0.499999999999999 -> NaN 0.599999999999999 -> NaN 0.699999999999998 -> NaN 0.799999999999998 -> NaN 0.899999999999998 -> NaN 0.999999999999998 -> NaN 1.099999999999999 -> 1.299999999999996 1.2 -> 1.599999999999999 1.3 -> 1.900000000000001 1.400000000000001 -> 2.2 1.500000000000002 -> 2.500000000000001 1.600000000000003 -> 2.800000000000007 1.700000000000004 -> 3.100000000000014 1.800000000000005 -> 3.400000000000012 1.900000000000006 -> 3.700000000000002 2.000000000000004 -> 4.000000000000002 2.100000000000005 -> 4.500000000000003 2.200000000000006 -> 5.000000000000036 2.300000000000007 -> 5.500000000000036 2.400000000000001 -> 6.000000000000036 2.500000000000001 -> 6.500000000000004 2.600000000000001 -> 7.000000000000005 2.700000000000001 -> 7.500000000000005 2.800000000000001 -> 8.000000000000005 2.9000000000000012 -> 8.500000000000007

(таблица 1)

Задание выполнено

Лабораторная работа была выполнена.