

Лабораторная работа №3

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

Ход выполнения

Первым делом было прочитано ТЗ. На этот раз задания уже куда интересней пошли. Исключения – отдельная тема, которую не стоит пренебрегать лишний раз(очень много столкнулся с ними в работе за несколько лет в Java, да и во всех ЯП в принципе)

Задание 1

Мной были прочитаны данные источники:

<https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Exception.html>

<https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/IndexOutOfBoundsException.html>

<https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/ArrayIndexOutOfBoundsException.html>

<https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/IllegalArgumentException.html>

<https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/IllegalStateException.html>

Можно обнаружить тот факт, что все они являются Throwable, Serializable. Также часть ошибок может быть выброшена именно во время выполнения(и как ни странно, по опыту – их необязательно обрабатывать(компилятор позволит), но надо быть готовым к тому, что приложение сломается)

Соответственно, прочитав немного наперед, сформируем таблицу 1, содержащий некоторую важную инфу

Название	Когда кидается	Является ли runtime?
Exception	Просто ошибка. Любая. Это базовый класс, от которого были созданы все остальные ошибки	Нет
IndexOutOfBoundsException	Индекс лежит вне определенного диапазона	Да
ArrayIndexOutOfBoundsException	Попытка работать с массивом через “неправильные индексы”	Да
IllegalArgumentException	Неподходящие аргументы для метода(к примеру, отрицательные числа для функции корня)	Да
IllegalStateException	Метод вызывается в “неподходящее” время	Да

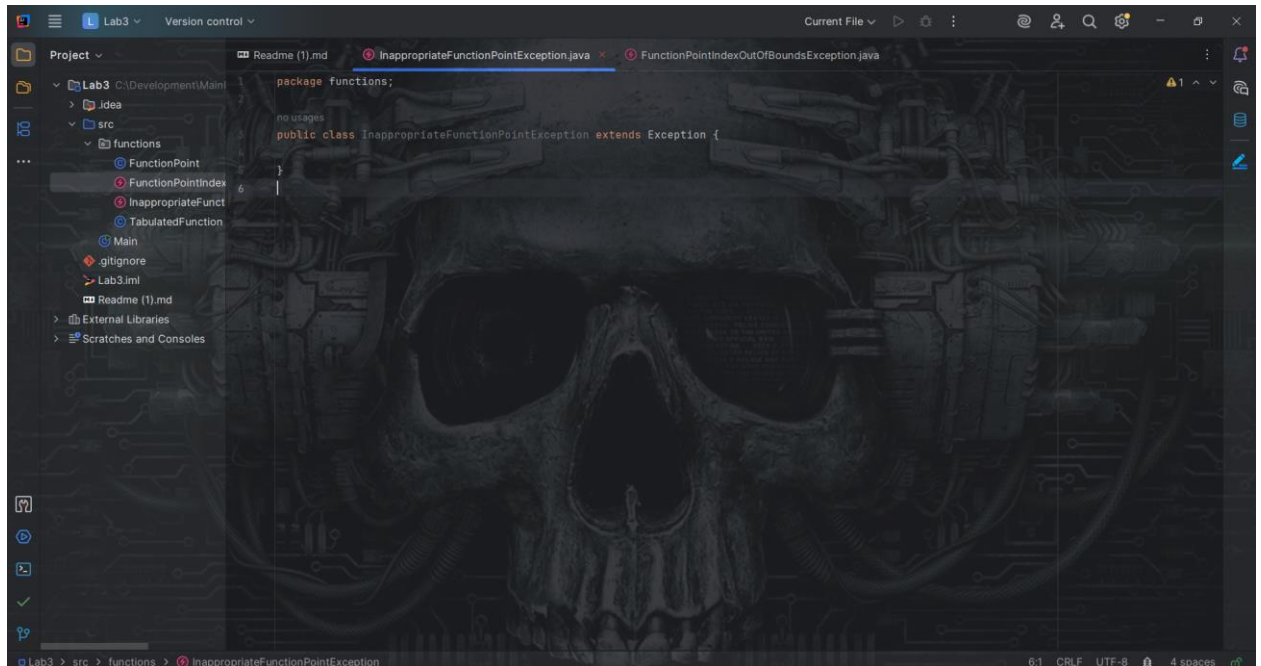
(таблица 1)

Т.к. я сижу на JDK 21, то и соответственно, ищу информацию в соответствии с версией Java

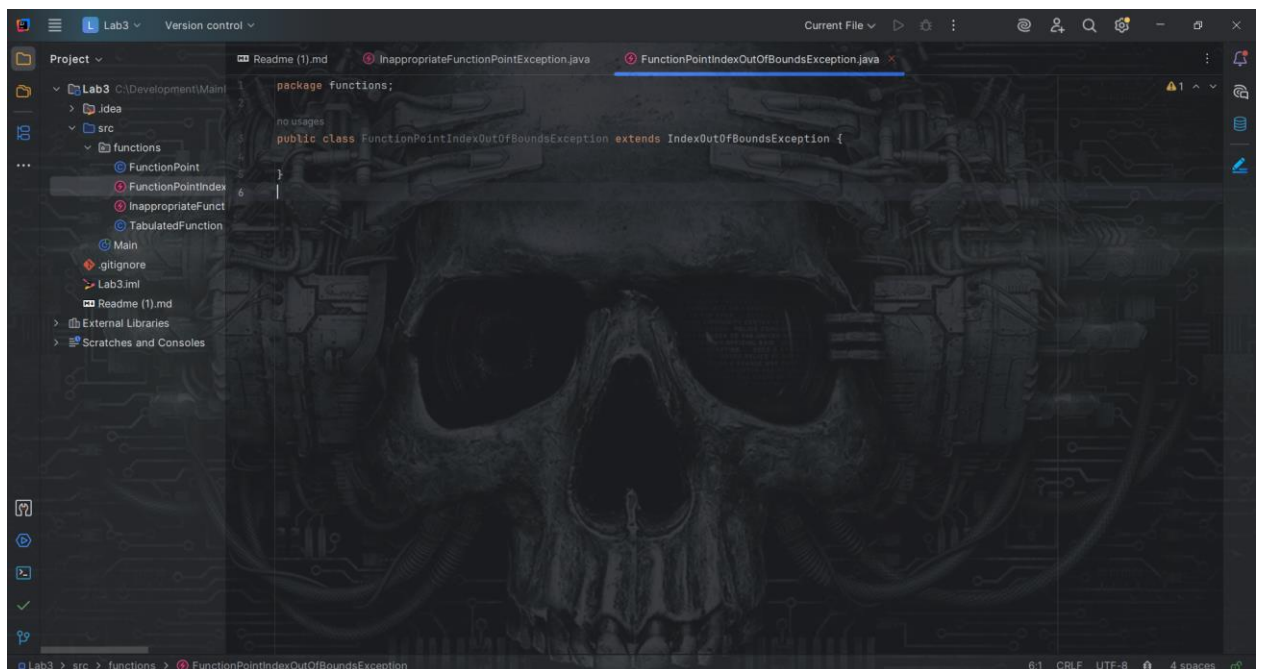
Задание выполнено

Задание 2

Следующей моей задачей было создание двух классов, наследующих определенные ошибки. Результат можем взглянуть на скрине 1-2(IntelliJ IDEA очень красиво их обозначил)



(скрин 1)



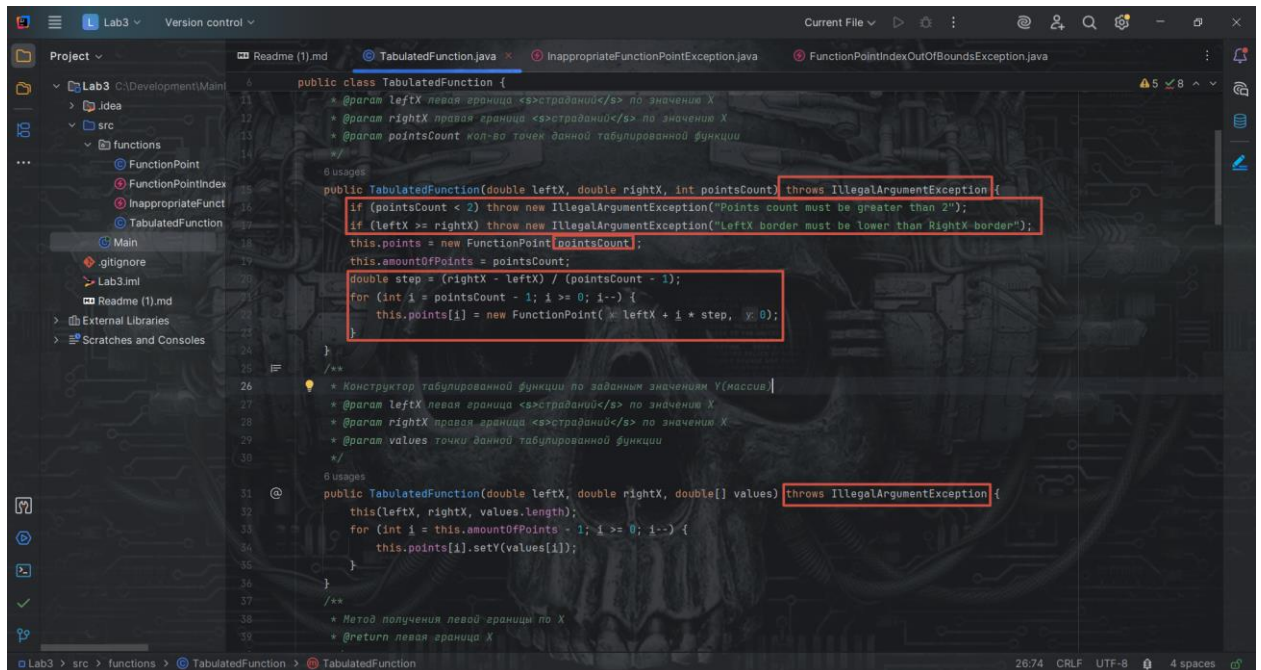
(скрин 2)

Т.к. по ТЗ их не нужно было заполнять, так что оставил их такими

Задание выполнено

Задание 3

Следующей моей задачей была модификация класса TabulatedFunction. Вносим правки...(см скрины 3-8). Красными прямоугольниками обозначены зоны модификации(что-то удалил, изменил, вставил). Возможно что-то могу упустить

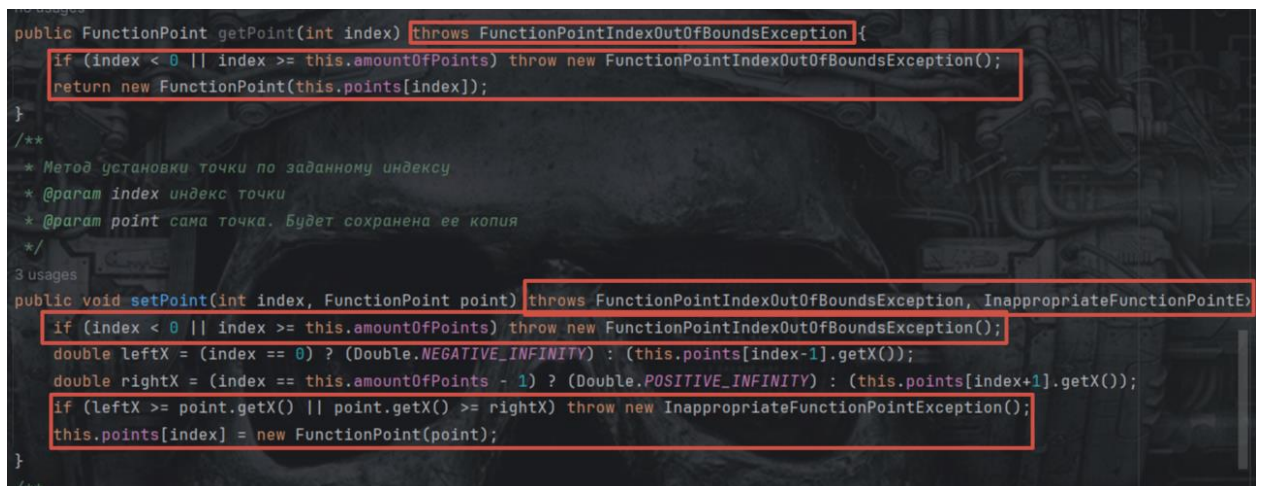


```
public class TabulatedFunction {
    /**
     * @param leftX левая граница <math>\langle x \rangle</math> по значению X
     * @param rightX правая граница <math>\langle x \rangle</math> по значению X
     * @param pointsCount кол-во точек данной табулированной функции
     */
    // 0 usages
    public TabulatedFunction(double leftX, double rightX, int pointsCount) throws IllegalArgumentException {
        if (pointsCount < 2) throw new IllegalArgumentException("Points count must be greater than 2");
        if (leftX >= rightX) throw new IllegalArgumentException("LeftX border must be lower than RightX border");
        this.points = new FunctionPoint[pointsCount];
        this.amountOfPoints = pointsCount;
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = pointsCount - 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 <math>\langle x \rangle</math> по значению X
     * @param rightX правая граница <math>\langle x \rangle</math> по значению X
     * @param values точки данной табулированной функции
     */
    // 0 usages
    public TabulatedFunction(double leftX, double rightX, double[] values) throws IllegalArgumentException {
        this(leftX, rightX, values.length);
        for (int i = this.amountOfPoints - 1; i >= 0; i--) {
            this.points[i].setY(values[i]);
        }
    }

    /**
     * Метод получения левой границы по X
     * @return левая граница X
     */
}
```

(скрин 3)



```
public FunctionPoint getPoint(int index) throws FunctionPointIndexOutOfBoundsException {
    if (index < 0 || index >= this.amountOfPoints) throw new FunctionPointIndexOutOfBoundsException();
    return new FunctionPoint(this.points[index]);
}

/**
 * Метод установки точки по заданному индексу
 * @param index индекс точки
 * @param point сама точка. Будет сохранена ее копия
 */
// 3 usages
public void setPoint(int index, FunctionPoint point) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException {
    if (index < 0 || index >= this.amountOfPoints) throw new FunctionPointIndexOutOfBoundsException();
    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) throw new InappropriateFunctionPointException();
    this.points[index] = new FunctionPoint(point);
}

/**
 */
}
```

(скрин 4)


```

no usages
public double getPointX(int index) throws FunctionPointIndexOutOfBoundsException {
    if (index < 0 || index >= this.amountOfPoints) throw new FunctionPointIndexOutOfBoundsException();
    return this.points[index].getX();
}

/**
 * Метод изменения у точки X координаты
 * @param index индекс, чьей точки мы меняем X
 * @param x значение X координаты
 */
2 usages
public void setPointX(int index, double x) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException {
    if (index < 0 || index >= this.amountOfPoints) throw new FunctionPointIndexOutOfBoundsException();
    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) throws FunctionPointIndexOutOfBoundsException {
    if (index < 0 || index >= this.amountOfPoints) throw new FunctionPointIndexOutOfBoundsException();
    return this.points[index].getY();
}

```

(скрин 5, здесь все изменено, не видел смысла выделять)

```

/**
 * 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) throws FunctionPointIndexOutOfBoundsException {
    if (index < 0 || index >= this.amountOfPoints) throw new FunctionPointIndexOutOfBoundsException();
    this.points[index].setY(y);
}

/**
 * Метод удаления точки по заданному индексу
 * @param index индекс удаляемой точки
 */
1 usage
public void deletePoint(int index) throws FunctionPointIndexOutOfBoundsException {
    if (index < 0 || index >= this.amountOfPoints) throw new FunctionPointIndexOutOfBoundsException();
    for (; index < this.amountOfPoints - 2; index++) {
        this.points[index] = this.points[index + 1]; // делаем циклический сдвиг
    }
    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 6)

```

/**
 * Метод удаления точки по заданному индексу
 * @param index индекс удаляемой точки
 */
1 usage
public void deletePoint(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException {
    if (index < 0 || index >= this.amountOfPoints) throw new FunctionPointIndexOutOfBoundsException();
    if (this.amountOfPoints < 3) throw new IllegalArgumentException("Points count must be greater than 2 for deletion");
    for (; index < this.amountOfPoints - 2; index++) {
        this.points[index] = this.points[index + 1]; // делаем циклический сдвиг
    }
    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 7, упустил добавление еще одной ошибки)

```

/**
 * Метод добавления точки в список
 * @param point сама собственно точка
 */
1 usage
public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
    double x = point.getX();
    int index = 0;
    for (; index < this.amountOfPoints; index++) {
        if (this.points[index].getX() == x) throw new InappropriateFunctionPointException();
        if (this.points[index].getX() > x) break;
    }
    if (this.points.length == this.amountOfPoints) {
        FunctionPoint[] newArray = new FunctionPoint[this.points.length * 2];
        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);
        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 8)

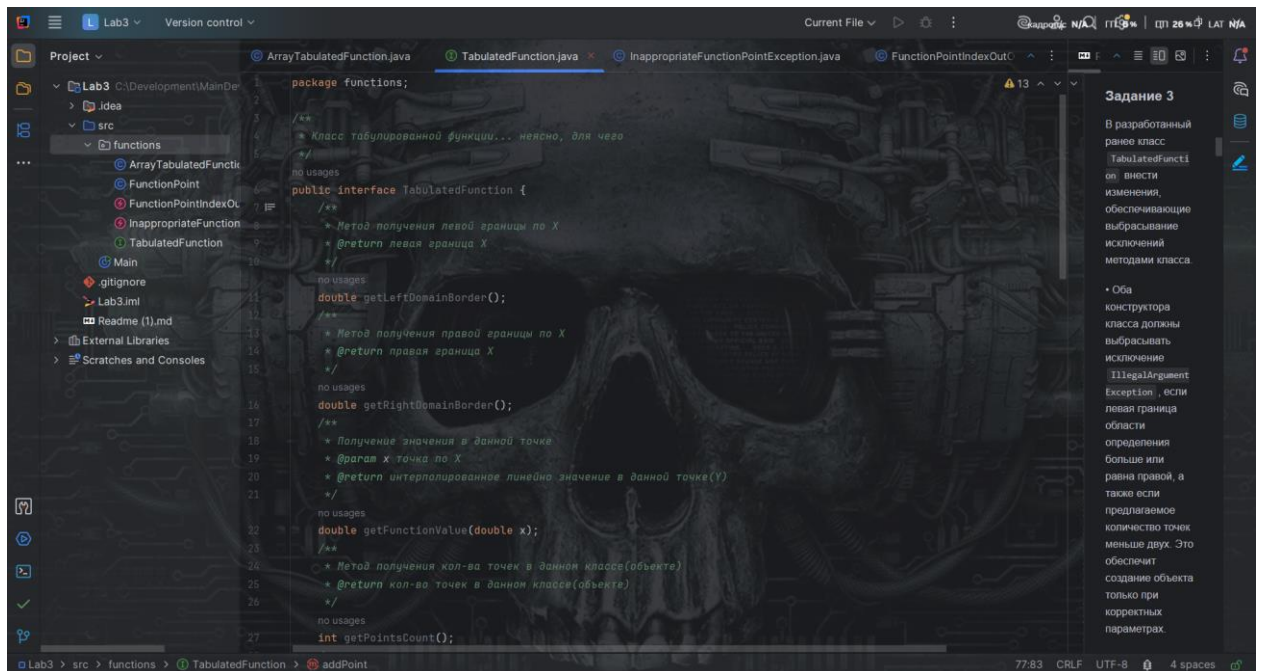
Задание выполнено

А теперь серьезно. Я прочитал задание 4, 5, 6, заметил, что ваша реализация порядка... ну как то не очень. Я начну с 6 задания, т.к. целесообразно сначала сделать интерфейс

Задание 6

Соответственно TabulatedFunction я переименовал в ArrayTabulatedFunction. Скопировал файл, назвал в TabulatedFunction обратно, сделал его интерфейсом, убрал конструкторы, тела методов(см скрин 9 для общего представления). Также удалю все документации к

методам в ArrayTabulatedFunction, сделаю так, чтобы он имплементировал интерфейс TabulatedFunction. А также расставил везде аннотации @Override(см скрин 10-11)



```
package functions;

/**
 * Класс табулированной функции... неясно, для чего
 */
public interface TabulatedFunction {

    /**
     * Метод получения левой границы по X
     * @return левая граница X
     */
    double getLeftDomainBorder();

    /**
     * Метод получения правой границы по X
     * @return правая граница X
     */
    double getRightDomainBorder();

    /**
     * Получение значения в данной точке
     * @param x точка по X
     * @return интерполированное линейно значение в данной точке(Y)
     */
    double getFunctionValue(double x);

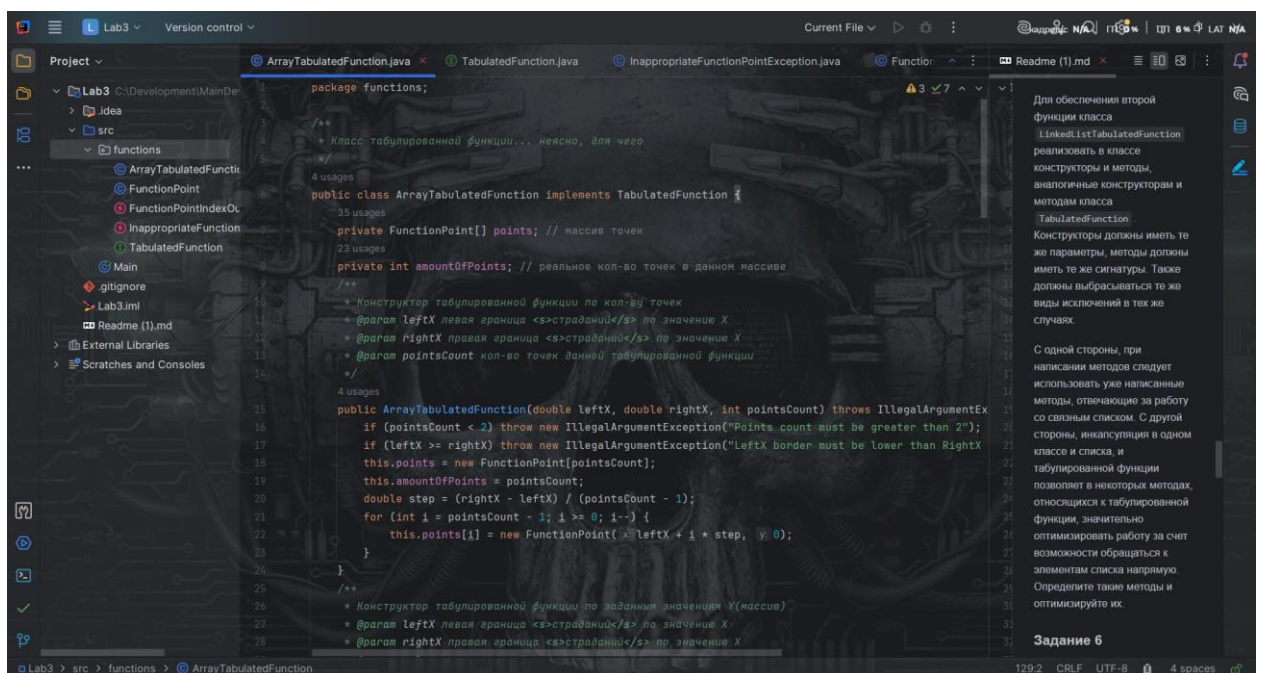
    /**
     * Метод получения кол-ва точек в данном классе(объекте)
     * @return кол-во точек в данном классе(объекте)
     */
    int getPointsCount();
}
```

Задание 3

В разработанный ранее класс TabulatedFunction внести изменения, обеспечивающие выбрасывание исключений методами класса

- Оба конструктора класса должны выбрасывать исключение IllegalArgumentException, если левая граница области определения больше или равна правой, а также если предполагаемое количество точек меньше двух. Это обеспечит создание объекта только при корректных параметрах.

(скрин 9)



```
package functions;

/**
 * Класс табулированной функции... неясно, для чего
 */
public class ArrayTabulatedFunction implements TabulatedFunction {

    private FunctionPoint[] points; // массив точек
    private int amountOfPoints; // реальное кол-во точек в данном массиве

    /**
     * Конструктор табулированной функции по кол-ву точек
     * @param leftX левая граница <=>страданий</s> по значению X
     * @param rightX правая граница <=>страданий</s> по значению X
     * @param pointsCount кол-во точек данной табулированной функции
     */
    public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) throws IllegalArgumentException {
        if (pointsCount < 2) throw new IllegalArgumentException("Points count must be greater than 2");
        if (leftX >= rightX) throw new IllegalArgumentException("LeftX border must be lower than RightX");
        this.points = new FunctionPoint(pointsCount);
        this.amountOfPoints = pointsCount;
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = pointsCount - 1; i >= 0; i--) {
            this.points[i] = new FunctionPoint(x: leftX + i * step, y: 0);
        }
    }

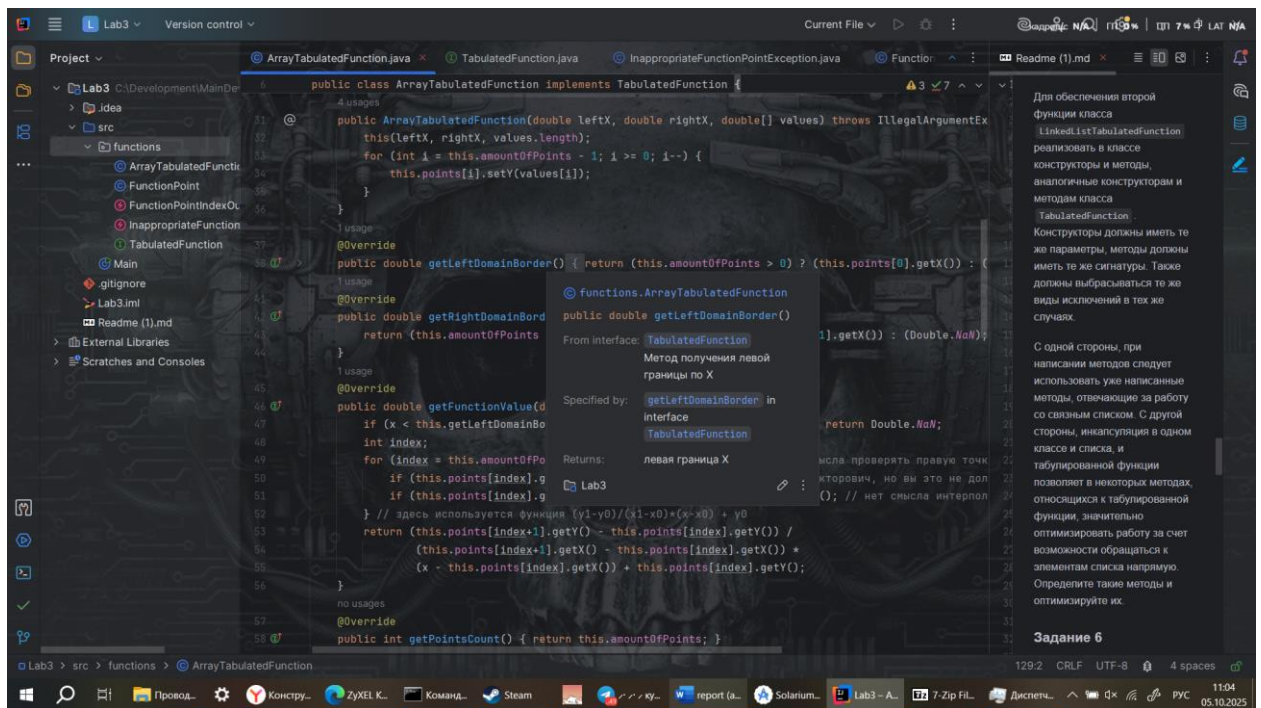
    /**
     * Конструктор табулированной функции по заданным значениям Y(массив)
     * @param leftX левая граница <=>страданий</s> по значению X
     * @param rightX правая граница <=>страданий</s> по значению X
     */
}
```

Задание 6

Для обеспечения второй функции класса LinkedTabulatedFunction реализовать в классе конструкторы и методы, аналогичные конструкторам и методам класса TabulatedFunction. Конструкторы должны иметь те же параметры, методы должны иметь те же сигнатуры. Также должны выбрасываться те же виды исключений в тех же случаях.

С одной стороны, при написании методов следует использовать уже написанные методы, отвечающие за работу со связным списком. С другой стороны, инкапсуляция в одном классе и списка, и табулированной функции позволяет в некоторых методах, относящихся к табулированной функции, значительно оптимизировать работу за счет возможности обращаться к элементам списка напрямую. Определите такие методы и оптимизируйте их.

(скрин 10)

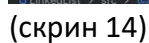
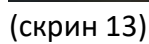


(скрин 11)

Задание выполнено

Задание 4-5

Теперь перейдем непосредственно к реализации нового класса – `LinkedListTabulatedFunction`. Его можно создать с помощью копирования от `ArrayTabulatedFunction`, но требует серьезного изменения всех методов, а также добавления внутреннего статического класса, реализующий `LinkedList` (к сожалению, легких путей не ищем). Начну с того, что организую отдельный проект, в котором напишу `LinkedList`, не имеющего предопределенного скажем так типа (т.е. дженерик). Протестирую список. Результаты вы можете видеть на скрине 11-14. Скрин 15 – я забыл добавить `onDelete()`, хотя можно и без этого обойтись



```

        this.tail.setNext(null);
        this.size--;
        return;
    }
    Node<T> temp = this.getNode(index);
    temp.getPrev().setNext(temp.getNext());
    temp.getNext().setPrev(temp.getPrev());
    temp.onDelete();
    this.size--;
}
1 usage
public void set(int index, T data) {

```

(скрин 15)

Протестирую все методы, которые написаны мною (отчет о тестировании не предоставляю, т.к. задание не подразумевало его же тестирование + прошлый семестр посвятили в их изучение и реализацию на C++). Список я решил реализовать двусвязный не-циклический, немного ускорил getNode тем, что он самостоятельно решает, с какой стороны начать (для меньшего кол-ва итераций). Середину не стал задавать, это и так ускорило примерно в 2 раза при худших случаях. Следующей задачей будет вставка данной реализации в LinkedListTabulatedFunction, а также серьезное изменение самого списка (поддержка инкапсуляции, задание типа FunctionPoint, вставка как внутренний статичный класс класса LinkedListTabulatedFunction). См проделанные изменения на скринах 16-17

```

public class LinkedListTabulatedFunction implements TabulatedFunction {
    public void addPoint(FunctionPoint point) throws InappropriateFunctionException {
        this.amountOfPoints++;
    }

    private static class LinkedList {
        private Node head = null;
        private Node tail = null;
        private int size = 0;

        public void addToEnd(FunctionPoint data) {
            if (head == null) {
                head = new Node(data);
                tail = head;
                this.size++;
            }
            this.tail.setNext(new Node(data));
            Node temp = this.tail;
            this.tail = this.tail.getNext();
            this.tail.setPrev(temp);
            this.size++;
        }

        private Node getNode(int index) {
            if (index < 0 || index >= this.size) throw new IndexOutOfBoundsException();
            if (this.size / 2 >= index) {

```

(скрин 16, впишу LinkedList в конец файла, убраю дженерик, где нужно – пропишу тип)


```

public FunctionPoint get() {
    return new FunctionPoint(this.data);
}

1 usage
public void set(FunctionPoint data) {
    this.data = new FunctionPoint(data);
}

```

(скрин 17, поддержка инкапсуляции в ноде. Списку плевать, что там в нодах)

Соответственно нужно внести изменения в самом классе LinkedListTabulatedFunction, добавить поддержку нового вложенного класса. Результаты см скрины 18-23

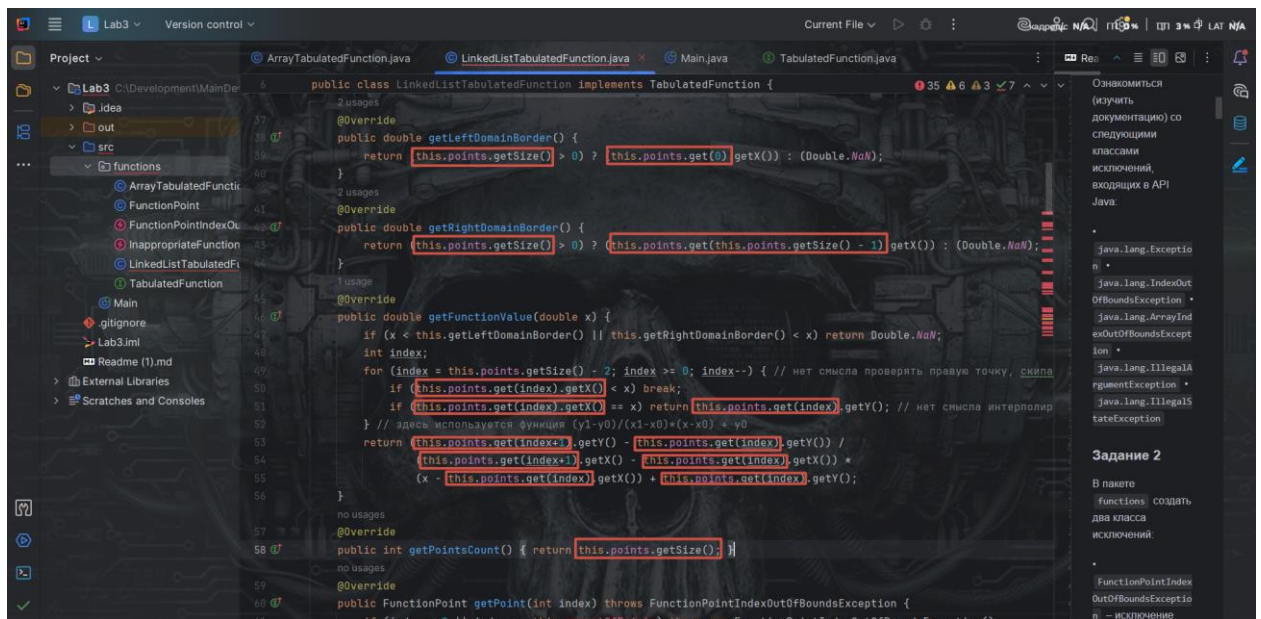
```

public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) throws IllegalArgumentException {
    if (pointsCount < 2) throw new IllegalArgumentException("Points count must be greater than 2");
    if (leftX >= rightX) throw new IllegalArgumentException("LeftX border must be lower than RightX border");
    this.points = new LinkedList();
    double step = (rightX - leftX) / (pointsCount - 1);
    for (int i = 0; i <= pointsCount - 1; i++) {
        this.points.addToEnd(new FunctionPoint(x: leftX + i * step, y: 0));
    }
}

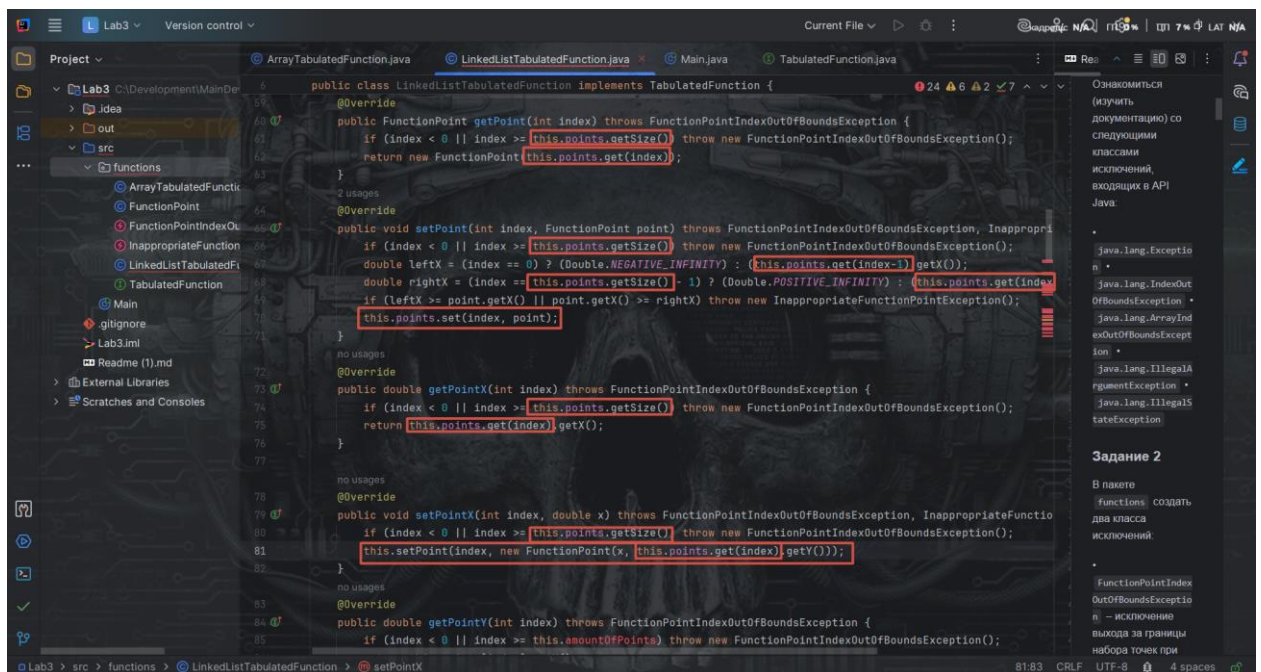
/**
 * Конструктор табулированной функции по заданным значениям Y(массив)
 * @param leftX левая граница <s>страданий</s> по значению X
 * @param rightX правая граница <s>страданий</s> по значению X
 * @param values точки данной табулированной функции
 */
11 usages
public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) throws IllegalArgumentException {
    this(leftX, rightX, values.length);
    for (int i = 0; i <= values.length - 1; i++) {
        FunctionPoint p = points.get(i);
        p.setY(values[i]);
        this.points.set(i, p);
    }
}

```

(скрин 19)



(скрин 20)



(скрин 21)

```

public double getPointY(int index) throws FunctionPointIndexOutOfBoundsException {
    if (index < 0 || index >= this.points.getSize()) throw new FunctionPointIndexOutOfBoundsException();
    return this.points.get(index).getY();
}

no usages
@Override
public void setPointY(int index, double y) throws FunctionPointIndexOutOfBoundsException {
    if (index < 0 || index >= this.points.getSize()) throw new FunctionPointIndexOutOfBoundsException();
    this.points.set(index, new FunctionPoint(this.points.get(index).getY(), y));
}

no usages
@Override
public void deletePoint(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException {
    if (index < 0 || index >= this.points.getSize()) throw new FunctionPointIndexOutOfBoundsException();
    if (this.points.getSize() < 3) throw new IllegalArgumentException("Points count must be greater than 2 for");
    this.points.remove(index);
}

```

(скрин 22)

```

@Override
public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
    double x = point.getX();
    int index = 0;
    for (; index < this.points.getSize(); index++) {
        if (this.points.get(index).getX() == x) throw new InappropriateFunctionPointException();
        if (this.points.get(index).getX() > x) break;
    }
    this.points.insert(index, point);
}

```

(скрин 23)

Задания выполнены

Задание 7

В принципе, начинается самая занудная часть. Тестирование

Тестировать я буду оба списка, вместе, чтобы наглядно сравнивать результаты. Заодно, если какие-то баги были бы, то всплыли...

Соответственно, в Main расположу новые методы, старые отредактирую... см скрин 24


```

public class Main {
    1 usage
    public static final double deltaX = 0.1;
    1 usage
    public static final double initialX = -1;
    1 usage
    public static final double endX = 3;
    public static void main(String[] args) {
        double[] arr = {0,1,4,9,16,25,36,49,64,81,100,121};

        TabulatedFunction f = new ArrayTabulatedFunction( leftX: 0, rightX: 10, arr);
        TabulatedFunction f2 = new LinkedListTabulatedFunction( leftX: 0, rightX: 10, arr);

        display(f);
        System.out.println(f.getPointsCount());
        System.out.println(f.getLeftDomainBorder());
        System.out.println(f.getRightDomainBorder());
        System.out.println("=====");
        display(f2);
        System.out.println(f2.getPointsCount());
        System.out.println(f2.getLeftDomainBorder());
        System.out.println(f2.getRightDomainBorder());
    }
    2 usages
    public static void display(TabulatedFunction tf) {
        double currentX = initialX;
        for (; currentX <= endX; currentX += deltaX) {
            System.out.print("~(" + (Math.round(currentX * 100) / 100d) + ") -> ");
            System.out.println(!Double.isNaN(tf.getFunctionValue(currentX)) ? Math.round(tf.getFunctionValue(currentX)) : NaN);
        }
    }
}

```

(скрин 24)

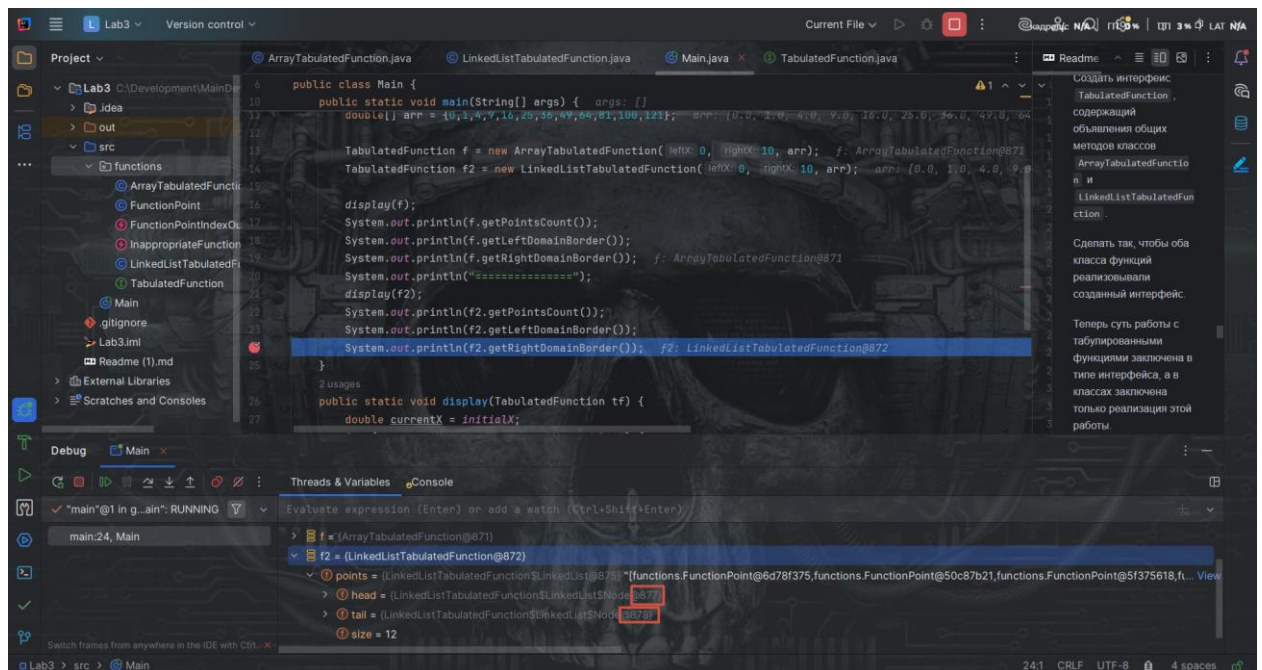
Запустим, посмотрим что выйдет(см таблица 2)

Array edition	LinkedList edition
~(-1.0) -> NaN	~(-1.0) -> NaN
~(-0.9) -> NaN	~(-0.9) -> NaN
~(-0.8) -> NaN	~(-0.8) -> NaN
~(-0.7) -> NaN	~(-0.7) -> NaN
~(-0.6) -> NaN	~(-0.6) -> NaN
~(-0.5) -> NaN	~(-0.5) -> NaN
~(-0.4) -> NaN	~(-0.4) -> NaN
~(-0.3) -> NaN	~(-0.3) -> NaN
~(-0.2) -> NaN	~(-0.2) -> NaN
~(-0.1) -> NaN	~(-0.1) -> NaN
~(0.0) -> NaN	~(0.0) -> NaN
~(0.1) -> 0.11	~(0.1) -> NaN
~(0.2) -> 0.22	~(0.2) -> NaN
~(0.3) -> 0.33	~(0.3) -> NaN
~(0.4) -> 0.44	~(0.4) -> NaN
~(0.5) -> 0.55	~(0.5) -> NaN
~(0.6) -> 0.66	~(0.6) -> NaN
~(0.7) -> 0.77	~(0.7) -> NaN
~(0.8) -> 0.88	~(0.8) -> NaN
~(0.9) -> 0.99	~(0.9) -> NaN
~(1.0) -> 1.3	~(1.0) -> NaN

~(1.1) -> 1.63	~(1.1) -> NaN
~(1.2) -> 1.96	~(1.2) -> NaN
~(1.3) -> 2.29	~(1.3) -> NaN
~(1.4) -> 2.62	~(1.4) -> NaN
~(1.5) -> 2.95	~(1.5) -> NaN
~(1.6) -> 3.28	~(1.6) -> NaN
~(1.7) -> 3.61	~(1.7) -> NaN
~(1.8) -> 3.94	~(1.8) -> NaN
~(1.9) -> 4.45	~(1.9) -> NaN
~(2.0) -> 5.0	~(2.0) -> NaN
~(2.1) -> 5.55	~(2.1) -> NaN
~(2.2) -> 6.1	~(2.2) -> NaN
~(2.3) -> 6.65	~(2.3) -> NaN
~(2.4) -> 7.2	~(2.4) -> NaN
~(2.5) -> 7.75	~(2.5) -> NaN
~(2.6) -> 8.3	~(2.6) -> NaN
~(2.7) -> 8.85	~(2.7) -> NaN
~(2.8) -> 9.56	~(2.8) -> NaN
~(2.9) -> 10.33	~(2.9) -> NaN
12	12
0.0	0.0
10.0	0.0

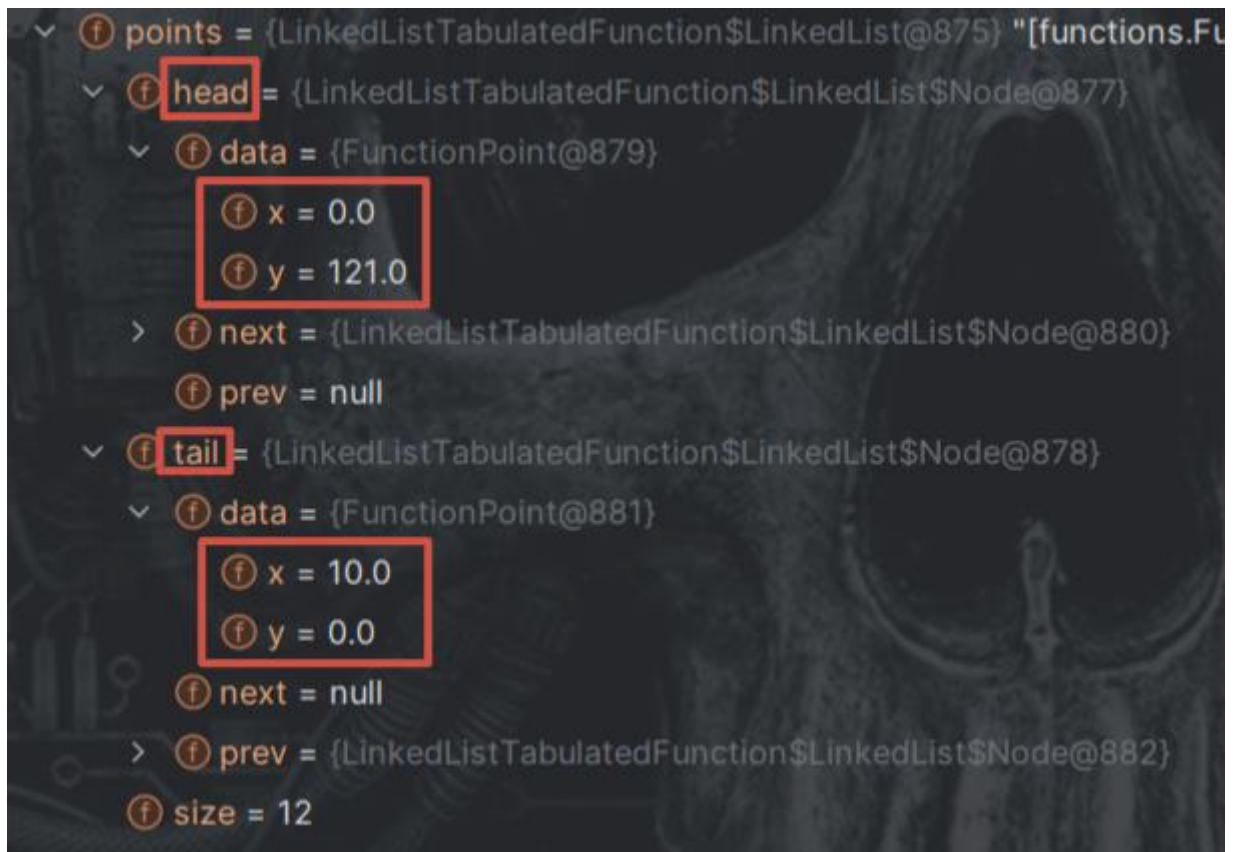
(таблица 2)

Видим, что какая-то фигня вышла. И скорее проблема в списке. Запустим Debugger, зададим точку останова после создания экземпляра класса LinkedListTabulatedFunction. Посмотрим это на скрине 25



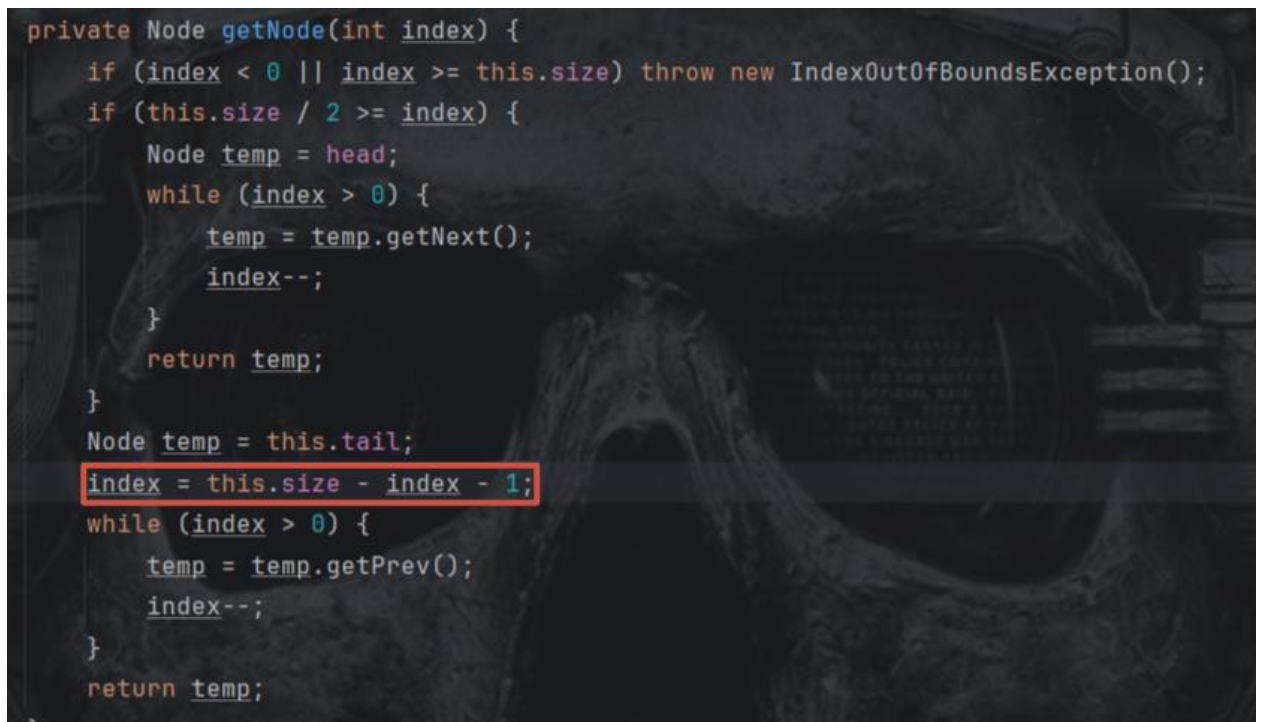
(скрин 25)

Из увиденного можно увидеть, что ссылки какие-то очень странные...(разница 1)
Проверим head и tail(скрин 26)



(скрин 26)

Данная картина совершенно не радует. Почему то порядком все попутано. Значит надо смотреть конструктор, а там из конструктора скорее делаем вывод о том, что проблема в получаемых нодах, т.к. вся остальная логика стабильная и нормальная. Вносим изменения, снова тестируем(см скрин 27 + таблица 3)



(скрин 27)

~(-1.0) -> NaN

~(-1.0) -> NaN

~(-0.9) -> NaN	~(-0.9) -> NaN
~(-0.8) -> NaN	~(-0.8) -> NaN
~(-0.7) -> NaN	~(-0.7) -> NaN
~(-0.6) -> NaN	~(-0.6) -> NaN
~(-0.5) -> NaN	~(-0.5) -> NaN
~(-0.4) -> NaN	~(-0.4) -> NaN
~(-0.3) -> NaN	~(-0.3) -> NaN
~(-0.2) -> NaN	~(-0.2) -> NaN
~(-0.1) -> NaN	~(-0.1) -> NaN
~(0.0) -> NaN	~(0.0) -> NaN
~(0.1) -> 0.11	~(0.1) -> 0.11
~(0.2) -> 0.22	~(0.2) -> 0.22
~(0.3) -> 0.33	~(0.3) -> 0.33
~(0.4) -> 0.44	~(0.4) -> 0.44
~(0.5) -> 0.55	~(0.5) -> 0.55
~(0.6) -> 0.66	~(0.6) -> 0.66
~(0.7) -> 0.77	~(0.7) -> 0.77
~(0.8) -> 0.88	~(0.8) -> 0.88
~(0.9) -> 0.99	~(0.9) -> 0.99
~(1.0) -> 1.3	~(1.0) -> 1.3
~(1.1) -> 1.63	~(1.1) -> 1.63
~(1.2) -> 1.96	~(1.2) -> 1.96
~(1.3) -> 2.29	~(1.3) -> 2.29
~(1.4) -> 2.62	~(1.4) -> 2.62
~(1.5) -> 2.95	~(1.5) -> 2.95
~(1.6) -> 3.28	~(1.6) -> 3.28
~(1.7) -> 3.61	~(1.7) -> 3.61
~(1.8) -> 3.94	~(1.8) -> 3.94
~(1.9) -> 4.45	~(1.9) -> 4.45
~(2.0) -> 5.0	~(2.0) -> 5.0
~(2.1) -> 5.55	~(2.1) -> 5.55
~(2.2) -> 6.1	~(2.2) -> 6.1
~(2.3) -> 6.65	~(2.3) -> 6.65
~(2.4) -> 7.2	~(2.4) -> 7.2
~(2.5) -> 7.75	~(2.5) -> 7.75
~(2.6) -> 8.3	~(2.6) -> 8.3
~(2.7) -> 8.85	~(2.7) -> 8.85
~(2.8) -> 9.56	~(2.8) -> 9.56
~(2.9) -> 10.33	~(2.9) -> 10.33
12	12
0.0	0.0
10.0	10.0

(таблица 3)

Косяк с конструктором исправлен. Данные совпадают полностью(столбцы такие же как и в таблице 2). Теперь протестируем базовые методы. Составим список(см таблица 4)

Номер	Метод
1	getLeftDomainBorder()
2	getRightDomainBorder()

3	getFunctionValue(double x)
4	getPointsCount()
5	getPoint(int index) throws FunctionPointIndexOutOfBoundsException
6	setPoint(int index, FunctionPoint point) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException
7	getPointX(int index) throws FunctionPointIndexOutOfBoundsException
8	setPointX(int index, double x) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException
9	getPointY(int index) throws FunctionPointIndexOutOfBoundsException
10	setPointY(int index, double y) throws FunctionPointIndexOutOfBoundsException
11	deletePoint(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException
12	addPoint(FunctionPoint point) throws InappropriateFunctionPointException

(таблица 4)

Протестировали получается 1-4. Проверим остальные. Напишем в Main нужные нам алгоритмы...(см скрин 28-29)

```
public static void main(String[] args) {
    double[] arr = {0,1,4,9,16,25,36,49,64,81,100,121};

    TabulatedFunction f = new ArrayTabulatedFunction( leftX: 0, rightX: 10, arr);
    TabulatedFunction f2 = new LinkedListTabulatedFunction( leftX: 0, rightX: 10, arr);

    test(f);
    System.out.println("=====");
    test(f2);
}

2 usages
public static void test(TabulatedFunction f) {
    System.out.println("Test 1");
    for (int i = 0; i >= -1; i--) {
        try {
            f.getPoint(i);
            System.out.println("Passed");
        } catch (FunctionPointIndexOutOfBoundsException err) {
            System.out.println("Right error");
        } catch (Exception err) {
            System.out.println("Another error");
        }
    }
    System.out.println("Test 2");
    for (int i = 0; i >= -1; i--) {
        try {
            f.setPoint(i, point: null);
        } catch (Exception err) {
            System.out.println("Another error");
        }
    }
}
```

(скрин 28)

```

}
System.out.println("Test 2");
for (int i = 0; i >= -1; i--) {
    try {
        f.setPoint(i, point: null);
        System.out.println("Passed");
    } catch (InappropriateFunctionPointException err) {
        System.out.println("Inappropriate error");
    } catch (FunctionPointIndexOutOfBoundsException err) {
        System.out.println("Right error");
    } catch (Exception err) {
        System.out.println("Another error");
    }
}
System.out.println("Test 3");
for (int i = 2; i >= -1; i--) {
    try {
        f.setPoint(i, new FunctionPoint( x: 0, y: 0));
        System.out.println("Passed");
    } catch (FunctionPointIndexOutOfBoundsException err) {
        System.out.println("IndexOf error");
    } catch (InappropriateFunctionPointException err) {
        System.out.println("Inappropriate error");
    } catch (Exception err) {
        System.out.println("Another error");
    }
}
}

```

(скрин 29)

Получили данные, записали в таблицу 5

Test 1	Test 1
Passed	Passed
Right error	Right error
Test 2	Test 2
Another error	Another error
Right error	Right error
Test 3	Test 3
Inappropriate error	Inappropriate error

Inappropriate error Passed IndexOf error	Inappropriate error Passed IndexOf error
--	--

(таблица 5)

Another error – скорее мы получили ошибку `NullPointerException`, ведь это и неудивительно – мы передавали `null`, а он как раз пытался у данного “объекта” методы. Остальные ошибки корректные. Теперь тестируем пункты 7-12 из таблицы 4. Изменим вновь `Main`(см скрины 30-33)

```
public static void main(String[] args) {
    double[] arr = {0,1,4,9,16,25,36,49,64,81,100,121};

    TabulatedFunction f = new ArrayTabulatedFunction( leftX: 0, rightX: 10, arr);
    TabulatedFunction f2 = new LinkedListTabulatedFunction( leftX: 0, rightX: 10, arr);

    display(f);
    System.out.println("=====");
    display(f2);

    System.out.println("=====");
    System.out.println("=====");
    System.out.println("=====");

    test(f);
    System.out.println("=====");
    test(f2);

    System.out.println("=====");
    System.out.println("=====");
    System.out.println("=====");
    f = new ArrayTabulatedFunction( leftX: 0, rightX: 10, arr); // жестокий способ отк
    f2 = new LinkedListTabulatedFunction( leftX: 0, rightX: 10, arr);

    test2(f);
    System.out.println("=====");
    test2(f2);
}
```

(скрин 30)

```
public static void test2(TabulatedFunction f) {  
    System.out.println("Test 1");  
    try {  
        f.getPointX( index: 1);  
        System.out.println("Passed");  
    } catch (Exception e) {  
        System.out.println("Non right error");  
    }  
    try {  
        f.getPointX( index: -1);  
        System.out.println("Passed");  
    } catch (FunctionPointIndexOutOfBoundsException e) {  
        System.out.println("Right error");  
    }  
    System.out.println("Test 2");  
    try {  
        f.setPointX( index: 0, x: 0);  
        System.out.println("Passed");  
    } catch (FunctionPointIndexOutOfBoundsException e) {  
        System.out.println("Index error");  
    } catch (InappropriateFunctionPointException e) {  
        System.out.println("Inappropriate error");  
    }  
    try {  
        f.setPointX( index: -1, x: 0);  
    } catch (FunctionPointIndexOutOfBoundsException e) {  
        System.out.println("Right error");  
    }  
}
```

(скрин 31)

```
} catch (FunctionPointIndexOutOfBoundsException e) {  
    System.out.println("Index error");  
} catch (InappropriateFunctionPointException e) {  
    System.out.println("Inappropriate error");  
}  
  
try {  
    f.setPointX( index: 0, x: 50);  
    System.out.println("Passed");  
} catch (FunctionPointIndexOutOfBoundsException e) {  
    System.out.println("Index error");  
} catch (InappropriateFunctionPointException e) {  
    System.out.println("Inappropriate error");  
}  
  
System.out.println("Test 3");  
try {  
    f.getPointY( index: 1);  
    System.out.println("Passed");  
} catch (Exception e) {  
    System.out.println("Non right error");  
}  
  
try {  
    f.getPointY( index: -1);  
    System.out.println("Passed");  
} catch (FunctionPointIndexOutOfBoundsException e) {  
    System.out.println("Right error");  
}  
}
```

(скрин 32)


```

        System.out.println("Passed");
    } catch (FunctionPointIndexOutOfBoundsException e)
        System.out.println("Right error");
    }
    System.out.println("Test 4");
    try {
        f.setPointY( index: 0, y: 0);
        System.out.println("Passed");
    } catch (FunctionPointIndexOutOfBoundsException e)
        System.out.println("Index error");
    }
    try {
        f.setPointY( index: -1, y: 0);
        System.out.println("Passed");
    } catch (FunctionPointIndexOutOfBoundsException e)
        System.out.println("Index error");
    }
}

```

(скрин 33)

Запишем полученные данные в таблицу 6

Test 1	Test 1
Passed	Passed
Right error	Right error
Test 2	Test 2
Passed	Passed
Index error	Index error
Inappropriate error	Inappropriate error
Test 3	Test 3
Passed	Passed
Right error	Right error
Test 4	Test 4
Passed	Passed
Index error	Index error

(таблица 6)

Все ошибки вышли, как мы и ожидали. Осталось последних два метода проверить. Напишем еще функцию, обзовем test3(), и еще test4(). См скрины 34-35

```
public static void test3(TabulatedFunction f) {
    System.out.println("Test 1");
    try {
        f.deletePoint( index: -1);
        System.out.println("Passed");
    } catch (FunctionPointIndexOutOfBoundsException e) {
        System.out.println("Index0f error");
    } catch (IllegalStateException e) {
        System.out.println("Illegal state error");
    }
    try {
        f.deletePoint( index: 0);
        System.out.println("Passed");
    } catch (FunctionPointIndexOutOfBoundsException e) {
        System.out.println("Index0f error");
    } catch (IllegalStateException e) {
        System.out.println("Illegal state error");
    }
    try {
        for (int i = 0; i < 20; i++) f.deletePoint( index: 0);
        System.out.println("Passed");
    } catch (FunctionPointIndexOutOfBoundsException e) {
        System.out.println("Index0f error");
    } catch (IllegalStateException e) {
        System.out.println("Illegal state error");
    }
}
```

(скрин 34)

```
public static void test4(TabulatedFunction f) {
    System.out.println("Test 1");
    try {
        f.addPoint(null);
        System.out.println("Passed");
    } catch (FunctionPointIndexOutOfBoundsException e) {
        System.out.println("Index0f error");
    } catch (InappropriateFunctionPointException e) {
        System.out.println("Inappropriate error");
    } catch (Exception e) {
        System.out.println("Another error");
    }
    try {
        f.addPoint(new FunctionPoint( x: 0, y: 0));
        System.out.println("Passed");
    } catch (FunctionPointIndexOutOfBoundsException e) {
        System.out.println("Index0f error");
    } catch (InappropriateFunctionPointException e) {
        System.out.println("Inappropriate error");
    } catch (Exception e) {
        System.out.println("Another error");
    }
    try {
        f.addPoint(new FunctionPoint( x: 500, y: 0));
        System.out.println("Passed");
    } catch (FunctionPointIndexOutOfBoundsException e) {
        System.out.println("Index0f error");
    } catch (InappropriateFunctionPointException e) {
        System.out.println("Inappropriate error");
    } catch (Exception e) {
        System.out.println("Another error");
    }
}
```

(скрин 35)

Внесем изменения в main(см скрин 36)


```

public static void main(String[] args) {
    System.out.println("=====");
    f = new ArrayTabulatedFunction( leftX: 0, rightX: 10, arr); // жестокий способ откатить данн
    f2 = new LinkedListTabulatedFunction( leftX: 0, rightX: 10, arr);

    test2(f);
    System.out.println("=====");
    test2(f2);

    System.out.println("=====");
    System.out.println("=====");
    System.out.println("=====");
    f = new ArrayTabulatedFunction( leftX: 0, rightX: 10, arr); // жестокий способ откатить данн
    f2 = new LinkedListTabulatedFunction( leftX: 0, rightX: 10, arr);

    test3(f);
    System.out.println("=====");
    test3(f2);

    System.out.println("=====");
    System.out.println("=====");
    System.out.println("=====");
    f = new ArrayTabulatedFunction( leftX: 0, rightX: 10, arr); // жестокий способ откатить данн
    f2 = new LinkedListTabulatedFunction( leftX: 0, rightX: 10, arr);

    test4(f);
    System.out.println("=====");
    test4(f2);
}

```

(скрин 36)

Запустим. Заметим, что появилась какая-то новая ошибка, что я даже не предусмотрел такую(см скрин 37)

```

Run  Main
Test 3
Passed
Right error
Test 4
Passed
Index error
=====
=====
=====
Test 1
Index error
Passed
Exception in thread "main" java.lang.IllegalArgumentException Create breakpoint : Points count must be greater than 2 for
    at functions.ArrayTabulatedFunction.deletePoint(ArrayTabulatedFunction.java:98)
    at Main.test3(Main.java:109)
    at Main.main(Main.java:43)
Process finished with exit code 1

```

(скрин 37)

Обратимся к данному методу, увидим, что по ошибке я вставил вместо State слово Argument... Переименуем ошибку в обоих классах. Запустим еще раз. Сформируем результат в таблицу 7

Test 1 IndexOf error Passed Illegal state error	Test 1 IndexOf error Passed Illegal state error
Test 1 Another error Inappropriate error Another error	Test 1 Another error Inappropriate error Passed

(таблица 7)

Итак, замечаем, что у табулированной функции версии Array есть не та ошибка, когда добавляем корректную точку. Давайте взглянем, что вызвало ее тогда... Изменим в test4() последнюю строку, написав e.printStackTrace(); (см скрин 38)

```

public static void test4(TabulatedFunction f) {
    System.out.println("IndexOf error");
} catch (InappropriateFunctionPointException e) {
    System.out.println("Inappropriate error");
} catch (Exception e) {
    System.out.println("Another error");
}
try {
    f.addPoint(new FunctionPoint( x: 0, y: 0));
    System.out.println("Passed");
} catch (FunctionPointIndexOutOfBoundsException e) {
    System.out.println("IndexOf error");
} catch (InappropriateFunctionPointException e) {
    System.out.println("Inappropriate error");
} catch (Exception e) {
    System.out.println("Another error");
}
try {
    f.addPoint(new FunctionPoint( x: 500, y: 0));
    System.out.println("Passed");
} catch (FunctionPointIndexOutOfBoundsException e) {
    System.out.println("IndexOf error");
} catch (InappropriateFunctionPointException e) {
    System.out.println("Inappropriate error");
} catch (Exception e) {
    e.printStackTrace();
}
}

```

(скрин 38)

Получим весьма неудовлетворительный результат...(см скрин 39)

```

java.lang.ArrayIndexOutOfBoundsException Create breakpoint Create breakpoint : arraycopy: length -2 is negative
at java.base/java.lang.System.arraycopy(Native Method)
at functions.ArrayTabulatedFunction.addPoint(ArrayTabulatedFunction.java:119)
at Main.test4(Main.java:80)
at Main.main(Main.java:53)

```

(скрин 39)

Внесем изменения(см скрин 40). Чисто технически, должен был скопировать оставшуюся часть... На всякий случай попробуем вывести размер получившихся списков/массивов(см скрин 41)

```
if (this.points.length == this.amountOfPoints) {
    FunctionPoint[] newArray = new FunctionPoint[this.points.length * 2];
    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);
    this.points = newArray; // по идее массив удалится, т.к. нету на него ссылок
} else {
```

(скрин 40)

```
test3(f);
System.out.println("=====");
test3(f2);

System.out.println("=====");
System.out.println("===== ADD");
System.out.println("=====");
f = new ArrayTabulatedFunction( leftX: 0, rightX: 10, arr); // жестокий способ откатить данные, но какой есть
f2 = new LinkedListTabulatedFunction( leftX: 0, rightX: 10, arr);

test4(f);
System.out.println("=====");
test4(f2);

System.out.println("=====");
System.out.println("===== size");
System.out.println("=====");

System.out.println(f.getPointsCount());
System.out.println("=====");
System.out.println(f2.getPointsCount());
```

(скрин 41)

Получим, что ошибка исправилась и размеры стали равными 13(т.к. изначально было 12(массив double)). Чисто технически, тестирование окончено, исправлены были даже ошибки из второй лабораторной работы.

Задание выполнено

p.s. – вы зачем то указали TabulatedFunctionImpl.java для ожидаемого результата, хотя сказали же вы создать интерфейс TabulatedFunction. Оставлю только второй вариант.

Лабораторная работа была выполнена.