

ЛАБОРАТОРНАЯ РАБОТА №3

Курс: Объектно-ориентированное
программирование.

Студент: Кузнецов Эрик Витальевич

Группа:6204-010302D

Преподаватель: Борисов Дмитрий Сергеевич

Задание 2

Ход выполнения работы:

- 1) В папке functions создаем файл `FunctionPointIndexOutOfBoundsException.java` и `InappropriateFunctionPointException.java`.
- 2) В классе наследуемся от `IndexOutOfBoundsException` и `Exception` соответственно.

3) Добавляем конструкторы по умолчанию

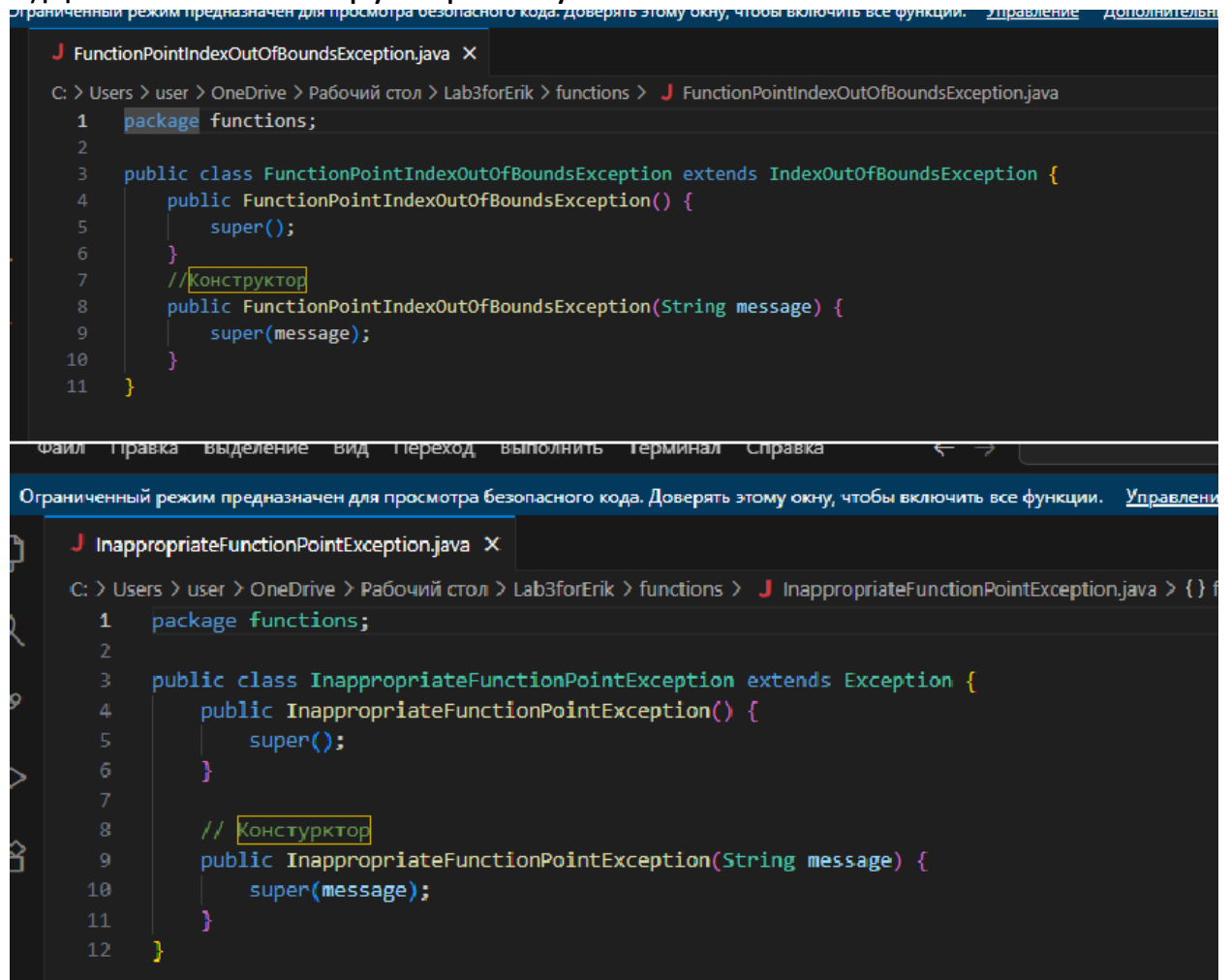


Рисунок 1,2-изображение с конечным результатом для задания 2.

Задание 3.

Ход выполнения работы:

- 1) В конструкторе `TabulatedFunction` проверяем условие `leftX >= rightX`. Если условие истинно, выбрасываем `IllegalArgumentException`
- 2) Добавляем проверку `pointsCount < 2`. Если условие истинно, выбрасываем `IllegalArgumentException`
- 3) В методах `getPoint()`, `setPoint()`, `getPointX()`, `setPointX()`, `getPointY()`, `setPointY()`, `deletePoint()` добавляем проверку индекса. Если `index < 0 || index >= pointsCount`, выбрасываем `FunctionPointIndexOutOfBoundsException`
- 4) В методах `setPoint()` и `setPointX()` проверяем соседние точки. Если новая координата `X` нарушает упорядоченность, выбрасываем `InappropriateFunctionPointException`
- 5) Модифицируем метод `addPoint()`: Добавляем проверку на существование точки с таким же `X`. Если точка с таким `X` существует, выбрасываем `InappropriateFunctionPointException`
- 6) Добавляем проверку `pointsCount < 3` в `deletePoint()`. Если условие истинно, выбрасываем `IllegalStateException`.

```
public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {
    if (leftX >= rightX) {
        throw new IllegalArgumentException("Левая граница должна быть меньше правой");
    }
    if (pointsCount < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не менее 2");
    }

    this.pointsCount = pointsCount;
    this.points = new FunctionPoint[pointsCount];
    double step = (rightX - leftX) / (pointsCount - 1);
    for (int i = 0; i < pointsCount; i++) {
        double x = leftX + i * step;
        points[i] = new FunctionPoint(x, 0);
    }
}

public ArrayTabulatedFunction(double leftX, double rightX, double[] values) {
    if (leftX >= rightX) {
        throw new IllegalArgumentException("Левая граница должна быть меньше правой");
    }
    if (values.length < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не менее 2");
    }

    this.pointsCount = values.length;
    this.points = new FunctionPoint[pointsCount];
    double step = (rightX - leftX) / (pointsCount - 1);
    for (int i = 0; i < pointsCount; i++) {
        double x = leftX + i * step;
        points[i] = new FunctionPoint(x, values[i]);
    }
}
```

```

public FunctionPoint getPoint(int index) {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " выходит за границы [0, " + (pointsCount-1) + "]");
    }
    return new FunctionPoint(points[index]);
}

public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException{
    // Проверка выхода за границы
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " выходит за границы [0, " + (pointsCount-1) + "]");
    }

    // Проверка упорядоченности и выброс исключения при нарушении
    if (index > 0 && point.getX() <= points[index - 1].getX()) {
        throw new InappropriateFunctionPointException("X координата точки нарушает упорядоченность с предыдущей точкой");
    }
    if (index < pointsCount - 1 && point.getX() >= points[index + 1].getX()) {
        throw new InappropriateFunctionPointException("X координата точки нарушает упорядоченность со следующей точкой");
    }

    points[index] = new FunctionPoint(point);
}

public double getPointX(int index) {
    // Проверка выхода за границы
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " выходит за границы [0, " + (pointsCount-1) + "]");
    }
    return points[index].getX();
}

public void setPointX(int index, double x) throws InappropriateFunctionPointException {
    // Проверка выхода за границы
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " выходит за границы [0, " + (pointsCount-1) + "]");
    }

    // Проверка упорядоченности и выброс исключения при нарушении
    if (index > 0 && x <= points[index - 1].getX()) {
        throw new InappropriateFunctionPointException("X координата точки нарушает упорядоченность с предыдущей точкой");
    }
    if (index < pointsCount - 1 && x >= points[index + 1].getX()) {
        throw new InappropriateFunctionPointException("X координата точки нарушает упорядоченность со следующей точкой");
    }
}

```

```

public void deletePoint(int index) {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " выходит за границы [0, " + (pointsCount-1) + "]");
    }

    // Проверка на минимальное количество точек
    if (pointsCount < 3) {
        throw new IllegalStateException("Невозможно удалить точку: количество точек должно быть не менее 3");
    }
    for (int i = index; i < pointsCount - 1; i++) {
        points[i] = points[i + 1];
    }
    --pointsCount;
}

public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException{
    for (int i = 0; i < pointsCount; i++) { // Проверка на дубликат X и выход за границы
        if (point.getX() == points[i].getX()) {
            throw new InappropriateFunctionPointException("Точка с X=" + point.getX() + " уже существует");
        }
    }
    int count=0;
    while(count < pointsCount && point.getX() > points[count].getX()){//проверка текущего X объекта с X объекта point
        count++;
    }
    // Проверяем необходимость увеличения массива
    if (pointsCount >= points.length) {
        FunctionPoint[] newPoints = new FunctionPoint[points.length * 2];
        for (int i = 0; i < pointsCount; i++) {
            newPoints[i] = points[i];
        }
        points = newPoints;
    }
    // Сдвигаем элементы ,Выходит так что на месте point[count] и points[count+1] стоят два одинаковых элемента
    for (int i = pointsCount; i > count; i--) {
        points[i] = points[i - 1];
    }
    // Вставляем новую точку
    points[count] = new FunctionPoint(point);

    // увеличиваем счетчик
    pointsCount++;
}

```

Рисунок 3,4,5-Снимок Экрана с конечным результатом для задания 3

Задание 4

Ход выполнения работы:

- 1) В папке functions создаем файл LinkedListTabulatedFunction.java. Объявляем что класс реализует интерфейс TabulatedFunction
- 2) Создаем внутренний класс элемента списка и объявляем статический приватный класс ListElement. Добавляем такие поля, как: FunctionPoint point, ListElement prev, ListElement next
- 3) Создаем конструктор с параметром FunctionPoint. Объявляем поля основного класса. Добавляем поле head типа ListElement. Добавляем поле size типа int
- 4) Реализуем конструктор по умолчанию, после создаем объект head с null в качестве точки. Устанавливаем head.prev = head и head.next = head и инициализируем size = 0.
- 5) Реализуем метод getNodeByIndex (). Проверяем корректность индекса. Если индекс в первой половине списка, идем с начала. Если индекс во второй половине списка, идем с конца. Возвращаем найденный элемент
- 6) Реализуем метод addNodeToTail (). Создаем новый элемент и находим последний элемент через head.prev. Перестраиваем связи между последним, новым и head и увеличиваем size и возвращаем новый элемент.
- 7) Реализуем метод addNodeByIndex (). Если индекс равен size, вызываем addNodeToTail (). Находим элемент по индексу и создаем новый элемент и вставляем перед найденным. Перестраиваем связи соседних элементов. Увеличиваем size и возвращаем новый элемент.
- 8) Реализуем метод deleteNodeByIndex (). Находим элемент по индексу. Перестраиваем связи соседних элементов, исключая удаляемый. Уменьшаем size и возвращаем удаленный элемент.

```

package functions;

public class LinkedListTabulatedFunction implements TabulatedFunction {

    private static class ListElement{
        private FunctionPoint point;
        private ListElement prev, next;
        private ListElement(FunctionPoint point){
            this.point = point;
        }
    }

    private final ListElement head;
    private int size;

    //Конструктор по умолчанию
    public LinkedListTabulatedFunction() {
        head = new ListElement(null);
        head.prev = head;
        head.next = head;
        size = 0;
    }

    public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount){
        this();
        if (leftX >= rightX){
            throw new IllegalArgumentException("Левая граница области определения должна быть меньше правой");
        }
        if (pointsCount < 2){
            throw new IllegalArgumentException("Количество точек табулирования не может быть меньше 2");
        }
        double step = (rightX - leftX) / (pointsCount - 1);
        for(int i = 0; i < pointsCount; i++){
            ListElement node = addNodeToTail();
            node.point = new FunctionPoint(leftX, 0);
            leftX += step;
        }
    }

    public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) {
        this();
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница области определения должна быть меньше правой");
        }
        if (values.length < 2) {
            throw new IllegalArgumentException("Количество точек табулирования не может быть меньше 2");
        }
    }
}

```

Рисунок 6-Снимок Экрана с конечным результатом для задания 4.

Задание 5

Ход выполнения работы:

1) Реализуем методы интерфейса:

`getLeftDomainBorder()` - возвращаем X первого элемента после head

`getRightDomainBorder()` - возвращаем X последнего элемента перед head

`getPointsCount()` - возвращаем size

`getPoint()` - используем `getNodeByIndex()` и возвращаем копию точки

`setPoint()` - проверяем упорядоченность и устанавливаем новую точку

2) Оптимизируем метод `addPoint()`. Обрабатываем особые случаи: пустой список, добавление в начало или конец. Для вставки в середину ищем позицию и используем `addNodeByIndex()`. Также проверяем на дубликат X перед вставкой.

Задание 6

Ход выполнения работы:

1) Меняем имя класса TabulatedFunction на ArrayTabulatedFunction. Добавляем implements TabulatedFunction

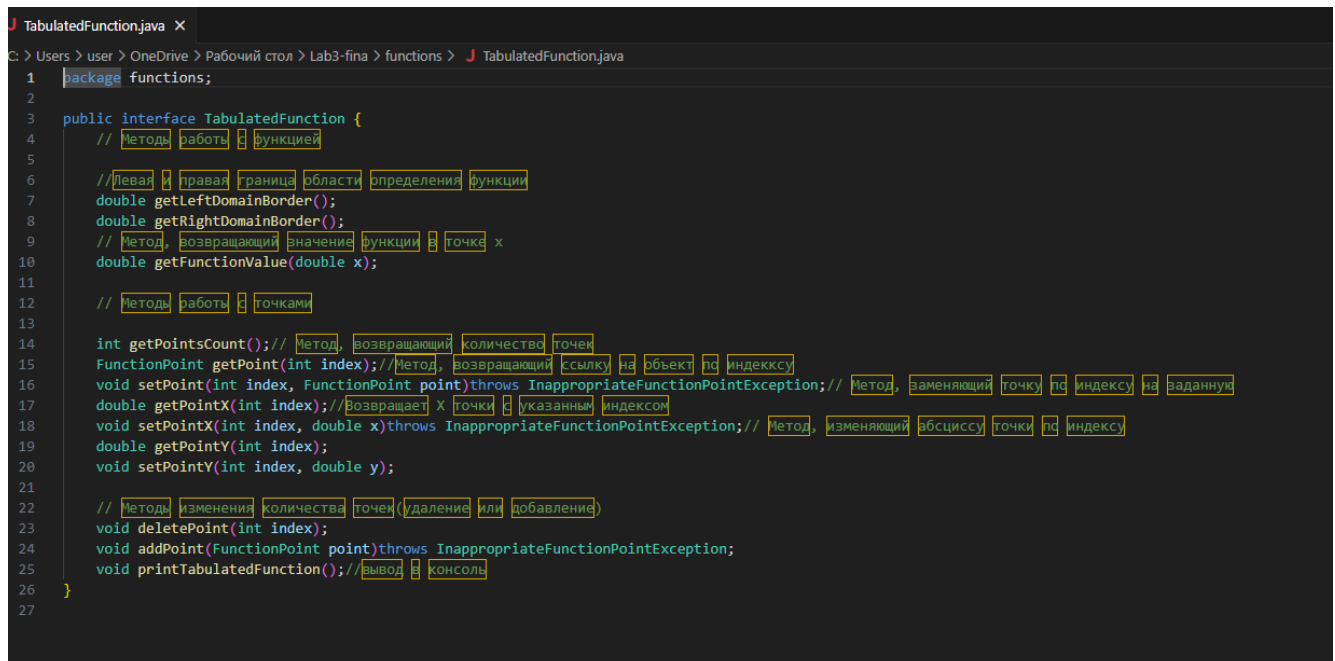
2) Создаем интерфейс:

В папке functions создаем файл TabulatedFunction.java.

Объявляем интерфейс с методами из обоих классов.

Указываем исключения в сигнатурах методов.

3) Реализуем интерфейс в классах: В ArrayTabulatedFunction добавляем implements TabulatedFunction. В LinkedListTabulatedFunction добавляем implements TabulatedFunction.



```
1 package functions;
2
3 public interface TabulatedFunction {
4     // Методы работы с функцией
5
6     // Левая и правая граница области определения функции
7     double getLeftDomainBorder();
8     double getRightDomainBorder();
9     // Метод, возвращающий значение функции в точке x
10    double getFunctionValue(double x);
11
12    // Методы работы с точками
13
14    int getPointsCount(); // Метод, возвращающий количество точек
15    FunctionPoint getPoint(int index); // Метод, возвращающий ссылку на объект по индексу
16    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException; // Метод, заменяющий точку по индексу на заданную
17    double getPointX(int index); // Возвращает x точки с указанным индексом
18    void setPointX(int index, double x) throws InappropriateFunctionPointException; // Метод, изменяющий абсциссу точки по индексу
19    double getPointY(int index);
20    void setPointY(int index, double y);
21
22    // Методы изменения количества точек (удаление или добавление)
23    void deletePoint(int index);
24    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
25    void printTabulatedFunction(); // вывод в консоль
26 }
27
```

Рисунок 7-Снимок Экрана с конечным результатом для задания 6.

Задание 7

Ход выполнения работы:

- 1) Демонстрируем замену реализации.
- 2) Тестируем различные исключения.
- 3) Запускаем одинаковые тесты для обеих реализаций (ArrayTabulatedFunction и LinkedListTabulatedFunction)

```
user@DESKTOP-3I0KT53 MINGW64 ~/OneDrive/Рабочий стол/Lab3-fina (master)
```

```
$ java Main
```

```
Создание функции  $y=2x+1$  на определенном интервале.
```

```
Вариант 1: ArrayTabulatedFunction
```

```
Вычисление значений функции в разных точках:
```

```
f(-1.0) = не определена (вне области определения)
```

```
f(0.0) = 1.0
```

```
f(2.0) = 5.0
```

```
f(4.0) = 9.0
```

```
f(6.6) = 14.2
```

```
f(8.0) = 17.0
```

```
f(10.0) = 21.0
```

```
f(12.0) = не определена (вне области определения)
```

```
Границы области определения:
```

```
Левая граница: 0.0
```

```
Правая граница: 10.0
```

```
Исходные точки функции:
```

```
Точка 0:(0.0 1.0)
```

```
Точка 1:(2.0 5.0)
```

```
Точка 2:(4.0 9.0)
```

```
Точка 3:(6.0 13.0)
```

```
Точка 4:(8.0 17.0)
```

```
Точка 5:(10.0 21.0)
```

```
2. Изменение точек функции:
```

```
Изменение первой точки (индекс 0) на (0,100):
```

```
Точка 0:(0.0 100.0)
```

```
Точка 1:(2.0 5.0)
```

```
Точка 2:(4.0 9.0)
```

```
Точка 3:(6.0 13.0)
```

```
Точка 4:(8.0 17.0)
```

```
Точка 5:(10.0 21.0)
```

```
3. Добавление новой точки (6.6, 9.9):
```

```
Количество точек до добавления: 6
```

```
Количество точек после добавления: 7
```

```
Новые точки функции после добавления:
```

```
Точка 0:(0.0 100.0)
```

```
Точка 1:(2.0 5.0)
```

```
Точка 2:(4.0 9.0)
```

```
Точка 3:(6.0 13.0)
```

```
Точка 4:(6.6 9.9)
```

```
Точка 5:(8.0 17.0)
```

```
Точка 6:(10.0 21.0)
```

```
4. Удаляем точку с индексом 3:
```

```
Точка 0:(0.0 100.0)
```

```
Точка 1:(2.0 5.0)
```

```
Точка 2:(4.0 9.0)
```

```
Точка 3:(6.6 9.9)
```

```
Точка 4:(8.0 17.0)
```

```
Точка 5:(10.0 21.0)
```

```
Точка 5:(10.0 21.0)

=== ПРОВЕРКА ИСКЛЮЧЕНИЙ ДЛЯ ArrayTabulatedFunction ===

Пытаемся получить доступ к координате по индексу 100
Поймано исключение: Индекс 100 выходит за границы [0, 5]

Пытаемся в абциссу точки с индексом 2 вставить 10
Поймано исключение: X координата точки нарушает упорядоченность со следующей точкой

Пытаемся создать точку с x=6.6,y=15
Поймано исключение: Точка с X=6.6 уже существует

Вариант 2: LinkedListTabulatedFunction

Исходные точки функции ( $y = 2x+1$ ):
Точка 0:(0.0 1.0)
Точка 1:(2.0 5.0)
Точка 2:(4.0 9.0)
Точка 3:(6.0 13.0)
Точка 4:(8.0 17.0)
Точка 5:(10.0 21.0)

=== ПРОВЕРКА ИСКЛЮЧЕНИЙ ДЛЯ LinkedListTabulatedFunction ===

Пытаемся получить доступ к координате по индексу 100
Поймано исключение: null

Пытаемся в абциссу точки с индексом 2 вставить 10
Поймано исключение: X координата точки нарушает упорядоченность

Пытаемся создать точку с x=4,y=20
Поймано исключение: Точка с X=4.0 уже существует
```

Рисунок 8,9-Снимок экрана с конечным результатом для задания 7.