Лабораторная работа №2

Выполнил: Магера Никита Алексеевич

Студент группы 6203-010302D

Ход выполнения

Задание 1

В рамках первого задания требуется создать пакет functions в папке src, в котором далее будут создаваться классы программы. (рис.1-2)

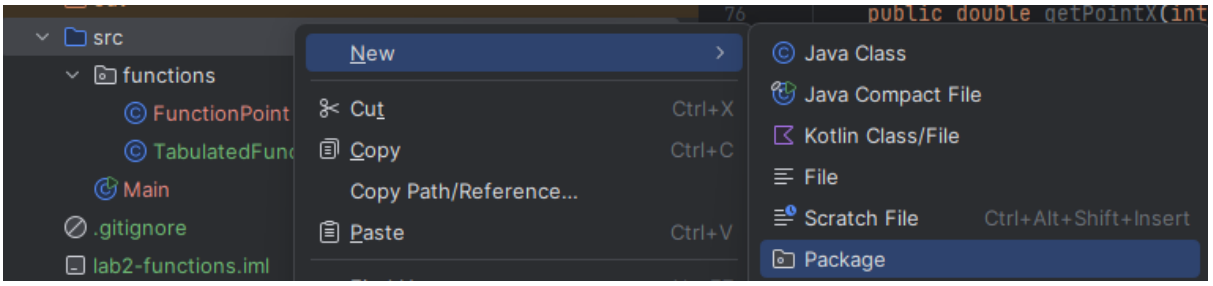


Рис. 1

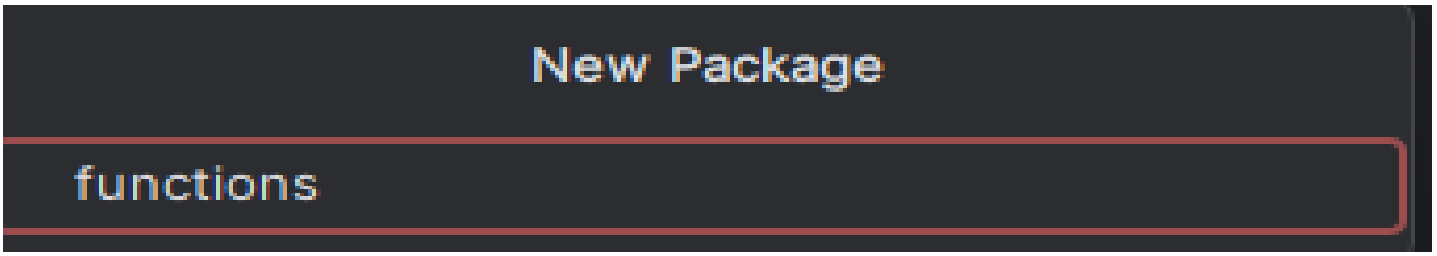


Рис. 2

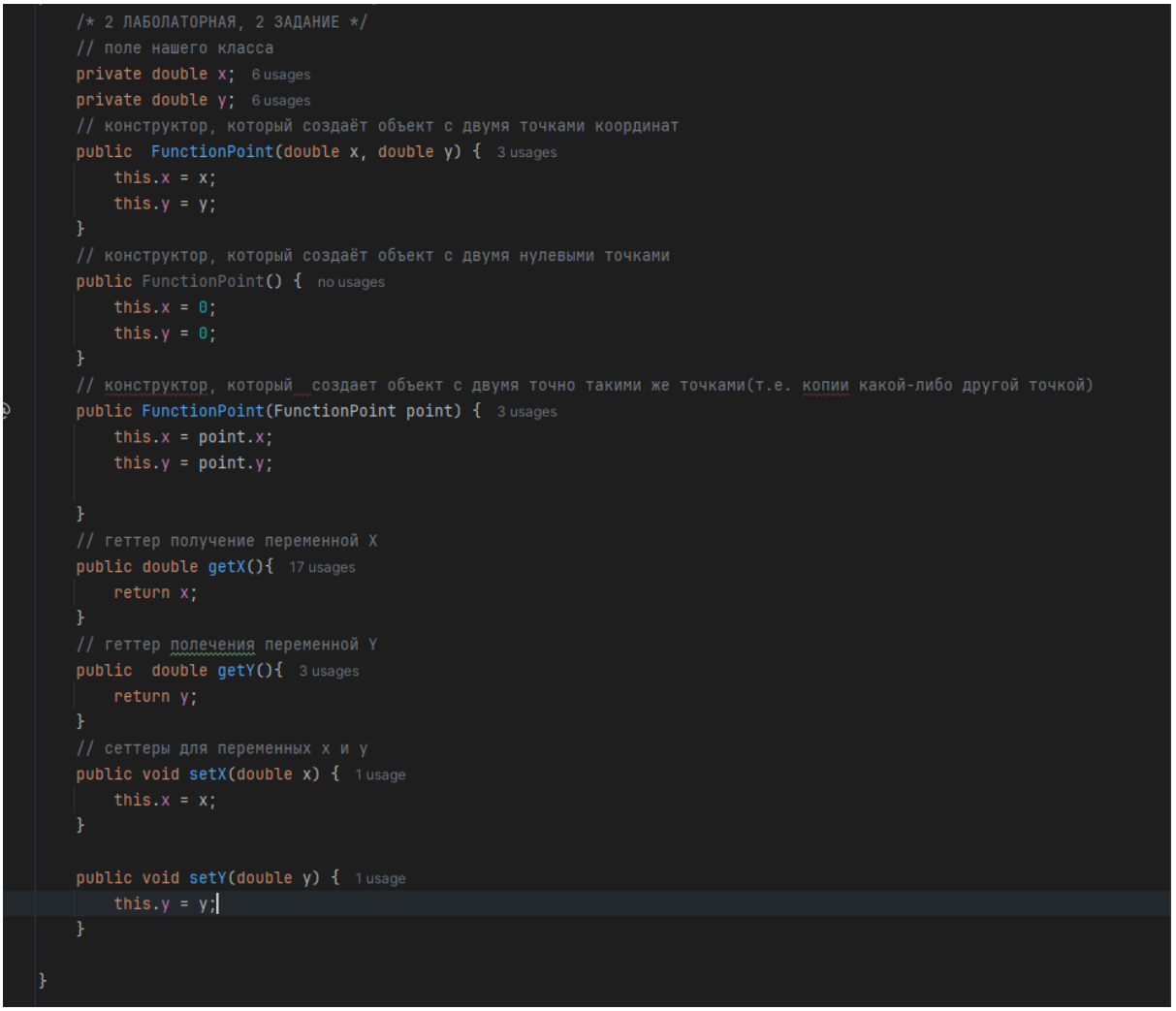
Задание 2

Во втором задании создал класс FunctionPoint для работы с точками функции. В этом классе есть две координаты - x и y, которые хранятся как приватные поля чтобы защитить данные.

Добавил три конструктора: первый FunctionPoint(double x, double y)- создает точку с заданными числами x и y;

второй FunctionPoint()- создает точку в нулевых координатах (0,0).

третий FunctionPoint(FunctionPoint point)-создает копию существующей точки. Также сделал методы getX и getY чтобы получать значения координат, и setX setY чтобы их менять. с помощью них нельзя просто так взять и поменять координаты точки.(рис.3)



```
    /* 2 ЛАБОЛАТОРНАЯ, 2 ЗАДАНИЕ */
    // поле нашего класса
    private double x;  6 usages
    private double y;  6 usages
    // конструктор, который создаёт объект с двумя точками координат
    public  FunctionPoint(double x, double y) {  3 usages
        this.x = x;
        this.y = y;
    }
    // конструктор, который создаёт объект с двумя нулевыми точками
    public FunctionPoint() {  no usages
        this.x = 0;
        this.y = 0;
    }
    // конструктор, который  создает объект с двумя точно такими же точками(т.е. копии какой-либо другой точкой)
    public FunctionPoint(FunctionPoint point) {  3 usages
        this.x = point.x;
        this.y = point.y;

    }
    // геттер получение переменной X
    public double getX(){  17 usages
        return x;
    }
    // геттер полечения переменной Y
    public  double getY(){  3 usages
        return y;
    }
    // сеттеры для переменных x и y
    public void setX(double x) {  1 usage
        this.x = x;
    }

    public void setY(double y) {  1 usage
        this.y = y;
    }

}
```

Рис.3

Задание 3

В третьем задании в пакете functions был создан класс TabulatedFunction, объект которого должен описывать табулированную функцию. Для хранения данных о точках должен использоваться массив типа FunctionPoint. При этом надо сделать массив так, чтобы точки в нём были всегда упорядочены по значению координаты x.

Также в этом классе должны быть описаны следующие конструкторы:

1) TabulatedFunction(double leftX, double rightX, int pointsCount) – создаёт объект табулированной функции по заданным левой и правой границе области определения, а также количеству точек для табулирования (значения функции в точках при этом следует считать равными 0);

2)TabulatedFunction(double leftX, double rightX, double[] values) – аналогичен предыдущему конструктору, но вместо количества точек получает значения функции в виде массива. В обоих конструкциях точки должны создаваться через равные интервалы по x. (рис.4)

```
public class TabulatedFunction {  3 usages  Nikitos100-15 *
    // создаем поле в виде массива
    private FunctionPoint[] points;  35 usages
    // для 6 задание поле количестов точек
    private int pointsCount;  17 usages

    // первый конструктор по заданию
    public TabulatedFunction(double leftX, double rightX, int pointsCount) {  no usages  Nikitos100-15 *
        this.points = new FunctionPoint[pointsCount]; // создаем объект для массива точек
        this.pointsCount = pointsCount; // инициализируем количество точек
        double step = (rightX - leftX) / (pointsCount - 1); // считаем шаг между точками
        for (int i = 0; i < pointsCount; i++) {
            double x_c = leftX + i * step;// считаем координату x
            points[i] = new FunctionPoint(x_c,  y: 0);// записываем наши точки в объект вида массива
        }
    }
    // второй конструктор по заданию
    public TabulatedFunction(double leftX, double rightX, double[] values) {  1 usage  Nikitos100-15 *
        this.points = new FunctionPoint[values.length]; // создаем объект для массива точек
        this.pointsCount = values.length; // инициализируем количество точек
        double step = (rightX - leftX) / (values.length - 1); // считаем шаг между точками
        for (int i = 0; i < values.length; i++) {
            double x_c = leftX + i * step; // считаем координату x
            points[i] = new FunctionPoint(x_c, values[i]); // записываем наши точки в объект вида массива
        }
    }

    /* 4 ЗАДАНИЕ */
```

Рис.4

Задание 4

В 4 задании добавил в класс TabulatedFunction методы для работы с табулированной функцией.

Метод getLeftDomainBorder- возвращает левую границу области определения функции. Это просто x координата самой первой точки в массиве, так как точки у нас всегда отсортированы по возрастанию x.

Метод getRightDomainBorder - возвращает правую границу области определения. Это x координата самой последней точки в массиве.

Метод getFunctionValue - получает значение функции в любой точке x с помощью линейной интерполяции. Если x находится за границами области определения (меньше левой границы или больше правой), метод возвращает Double.NaN - специальное значение "не число".

```java
// метод для получения значение функции в точке x, если эта точка лежит в области определения функции
public double getFunctionValue(double x) {  2 usages  Nikitos100-15 *
    // значение эпсилон как константа
    final double EPSILON = 1e-10;

    // проверка на пустоту и на существование(без нее у меня ломалось)
    if (points == null || pointsCount == 0) {
        return Double.NaN;
    }

    // получаем промежутки для проверки границ
    double leftX = points[0].getX();
    double rightX = points[pointsCount - 1].getX();

    // проверка границ с переменной эпсилон
    if ((x < (leftX - EPSILON)) || (x > (rightX + EPSILON))) {
        return Double.NaN;
    }

    for (int i = 0; i < pointsCount; i++) {
        if (Math.abs(x - points[i].getX()) < EPSILON) {
            return points[i].getY(); // возвращаем соответствующий y (также у меня не было вроде)
        }
    }

    // ищем интервал с учетом эпсилон для интерполяции
    for (int i = 0; i < pointsCount - 1; i++) {
        double x1 = points[i].getX();
        double x2 = points[i + 1].getX();

        // ищем значения x с учетом эпсилон
        if (x >= x1 - EPSILON && x <= x2 + EPSILON) {
            double y1 = points[i].getY();
            double y2 = points[i + 1].getY();
            return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
        }
    }

    return Double.NaN; // возвращаем на всякий случай
}
```

```java
/* 4 ЗАДАНИЕ */
// метод для возвращения самой левой границы области определения
public double getLeftDomainBorder() {  no usages  Nikitos100-15
    return points[0].getX();
}

// метод для возвращения самой правой границы области определения
public double getRightDomainBorder() {  no usages  Nikitos100-15 *
    return points[pointsCount - 1].getX();
}

// метод для получения значение функции в точке x, если эта точка лежит в области определения функции
public double getFunctionValue(double x) {  2 usages  Nikitos100-15 *
    // значение эпсилон как константа
    final double EPSILON = 1e-10;

    // проверка на пустоту и на существование(без нее у меня ломалось)
    if (points == null || pointsCount == 0) {
        return Double.NaN;
    }

    // получаем промежутки для проверки границ
    double leftX = points[0].getX();
    double rightX = points[pointsCount - 1].getX();

    // проверка границ с переменной эпсилон
    if ((x < (leftX - EPSILON)) || (x > (rightX + EPSILON))) {
        return Double.NaN;
    }

    for (int i = 0; i < pointsCount; i++) {
        if (Math.abs(x - points[i].getX()) < EPSILON) {
            return points[i].getY(); // возвращаем соответствующий y (также у меня не было вроде)
        }
```

Рис.5

Задание 5

В 5 задании добавил в класс TabulatedFunction методы для работы с отдельными точками функции.

Метод getPointsCount - возвращает общее количество точек в функции.

Метод getPoint возвращает копию точки по указанному индексу.

Метод setPoint заменяет точку на новую, но только если координата x новой точки находится между x соседних точек.

Методы getPointX и getPointY позволяют получить отдельно координату x или y точки по индексу.

Метод setPointX меняет координату x точки, но проверяет чтобы новое значение x оставалось между соседними точками.

Метод setPointY меняет координату y точки.(рис.6

```java
/* 5 ЗАДАНИЕ */
// метод для получения количества точек
public int getPointsCount() {  3 usages  new *
    return points.length;
}

// метод для получения копии точки
public FunctionPoint getPoint(int index) {  no usages  new *
    return new FunctionPoint(points[index]);
}

// метод для получения значения абциссы
public double getPointX(int index) {  no usages  new *
    return points[index].getX();
}

// метод для получения значения ординаты
public double getPointY(int index) {  no usages  new *
    return points[index].getY();
}

// метод для изменения значения ординаты точки с указанным номером
public void setPointY(int index, double y) {  1 usage  new *
    points[index].setY(y);
}

// метод для замены точки на табулированную
public void setPoint(int index, FunctionPoint point) {  no usages  new *
    // проверяем что новый x между соседними точками
    if (index > 0 && point.getX() <= points[index - 1].getX())
        return;
    if (index < points.length - 1 && point.getX() >= points[index + 1].getX())
        return;

    points[index] = new FunctionPoint(point);  // сохраняем копию
}
```

Рис.6

Задание 6

В предпоследнем пунтке добавил в класс TabulatedFunction методы для изменения количества точек.

Метод deletePoint удаляет точку по указанному индексу.

Метод addPoint добавляет новую точку в функцию.(рис.7)

```
/* 6 ЗАДАНИЕ */
// метод для удаления заданной точки
public void deletePoint(int index) {  1 usage  Nikitos100-15 *
    // сдвигаем все точки после удаляемой влево
    for (int i = index; i < pointsCount - 1; i++) {
        points[i] = points[i + 1];
    }
    pointsCount--;
}

public void addPoint(FunctionPoint point) {  1 usage  Nikitos100-15 *
    // проверяем, заполнен ли массив
    if (pointsCount >= points.length) {
        // увеличиваем массив в 2 раза
        FunctionPoint[] newPoints = new FunctionPoint[points.length * 2];
        System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, pointsCount);
        points = newPoints;
    }
    // ищем позицию для вставки x
    int pos = 0;
    while (pos < pointsCount && points[pos].getX() < point.getX()) {
        pos++;
    }
    // сдвигаем точки вправо, чтобы освободить место для новой точки
    System.arraycopy(points, pos, points, destPos: pos + 1, length: pointsCount - pos);
    // вставляем новую точку
    points[pos] = new FunctionPoint(point);
    pointsCount++;
}
```

Рис. 7

Задание 7

В последнем пункте создал класс Main вне пакета для тестирования табулированной функции. Взял линейную функцию f(x) = 2x + 1 на интервале [-2, 4].

Создал объект TabulatedFunction, передав массив значений Y. Вывел основные параметры функции - область определения и количество точек.

Проверил значения в разных точках: внутри области определения (x = -2, 0, 2, 4) и вне её (x = -4, 6). Для точек вне области определения получаем "не определено".

Протестировал модификацию функции: изменил координаты существующих точек, добавил новую точку (2.5, 6), удалил одну из точек. После каждой операции проверял количество точек и значения функции.

```java
import functions.*;
// переделанный меин 7 Задание
public class Main {  Nikitos100-15 *
    public static void main(String[] args) {  Nikitos100-15 *
        //создаем функцию f(x) = 2x + 1
        double[] xValues = {-2, -1, 0, 1, 2, 3, 4};
        double[] yValues = {-3, -1, 1, 3, 5, 7, 9};
        double leftX = xValues[0];
        double rightX = xValues[xValues.length - 1];
        TabulatedFunction linearFunc = new TabulatedFunction(leftX, rightX, yValues);

        // небольшая вводная часть
        System.out.println("f(x) = 2x + 1 на [" + leftX + ", " + rightX + "]");
        System.out.println("Точек: " + linearFunc.getPointsCount());

        // вывод всех точек
        System.out.println("Точки после создания функции:");
        printAllPoints(linearFunc);

        // тестируем значения
        System.out.println("Тест самих значений");
        double[] testPoints = {-4, -2, 0, 2, 4, 6};
        for (double x : testPoints) {
            double y = linearFunc.getFunctionValue(x);
            System.out.println("f(" + x + ") = " + y);
        }

        // меняем точки
        linearFunc.setPointY( index: 3,  y: 10);
        // точки после изменения y
        System.out.println("Точки после setPointY(3, 10):");
        printAllPoints(linearFunc);

        linearFunc.setPointX( index: 1,  x: -0.7);

        // вывод точки после изменения x
        System.out.println("Точки после setPointX(1, -0.7):");
        printAllPoints(linearFunc);
```

```java
    // добавляем и удаляем
    linearFunc.addPoint(new FunctionPoint( x: 2.5,  y: 6));

    // точки после добавления
    System.out.println("Точки после addPoint(2.5, 6):");
    printAllPoints(linearFunc);

    linearFunc.deletePoint( index: 2);
    // точки после удаления
    System.out.println("Точки после deletePoint(2):");
    printAllPoints(linearFunc);

    // финальная проверка
    System.out.println("Финальный результат:");
    System.out.println("Всего точек:" + linearFunc.getPointsCount());
    System.out.printf("f(2.5)=" + linearFunc.getFunctionValue( x: 2.5));
}

// метод для вывода всех точек через геттеры
public static void printAllPoints(TabulatedFunction linearFunc) {  5 usages  new *
    for (int i = 0; i < linearFunc.getPointsCount(); i++) {
        double x = linearFunc.getPointX(i);  // геттер для x
        double y = linearFunc.getPointY(i);  // геттер для y
        System.out.println("Точка "+ i +":" + "(" +x + "," + y +")");
    }
}
```

```
Точка 3:(1.0,10.0)
Точка 4:(2.0,5.0)
Точка 5:(3.0,7.0)
Точка 6:(4.0,9.0)
Точки после setPointX(1, -0.7):
Точка 0:(-2.0,-3.0)
Точка 1:(-0.7,-1.0)
Точка 2:(0.0,1.0)
Точка 3:(1.0,10.0)
Точка 4:(2.0,5.0)
Точка 5:(3.0,7.0)
Точка 6:(4.0,9.0)
Точки после addPoint(2.5, 6):
Точка 0:(-2.0,-3.0)
Точка 1:(-0.7,-1.0)
Точка 2:(0.0,1.0)
Точка 3:(1.0,10.0)
Точка 4:(2.0,5.0)
Точка 5:(2.5,6.0)
Точка 6:(3.0,7.0)
Точка 7:(4.0,9.0)
Точки после deletePoint(2):
Точка 0:(-2.0,-3.0)
Точка 1:(-0.7,-1.0)
Точка 2:(1.0,10.0)
Точка 3:(2.0,5.0)
Точка 4:(2.5,6.0)
Точка 5:(3.0,7.0)
Точка 6:(4.0,9.0)
Финальный результат:
Всего точек:7
f(2.5)=6.0
Process finished with exit code 0
```

Рис. 8-11