МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

ОТЧЕТ ПО
ЛАБОРАТОРНОЙ РАБОТЕ № 2

«Разработка набора классов для работы с табулированными функциями»

по курсу
Объектно-ориентированное программирование

Выполнил: Егорова-Екимкова Яна,
студент группы 6203-010302D

Оглавление

Задание №1

Создана директория «functions» с размещенными в ней файлами классов.

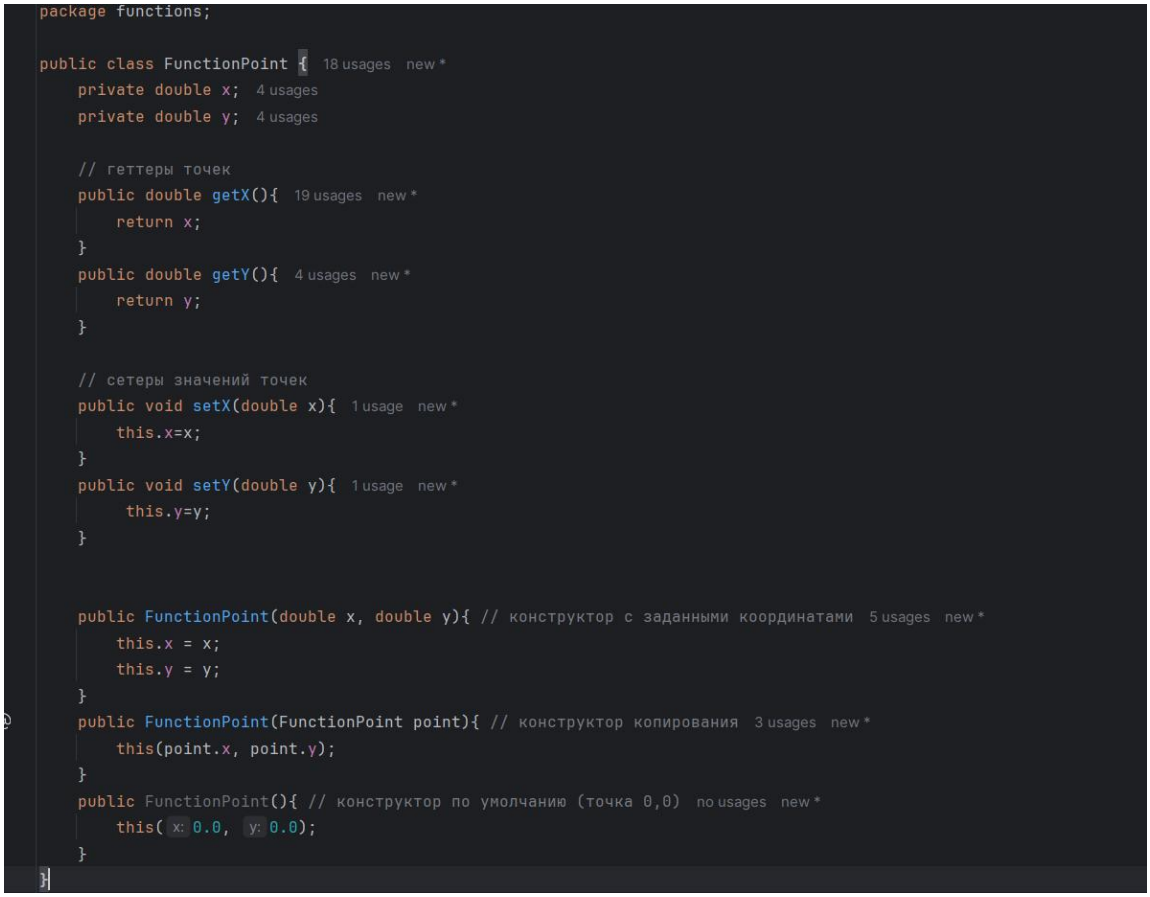
Задание №2

Для выполнения этого задания я разработала класс «FunctionPoint», представляющий отдельную точку функции с координатами (x, y). Класс написан с соблюдением принципов инкапсуляции поля объекта защищены от прямого доступа.

Мной были реализованы конструкторы:

- Инициализации с заданными координатами
- Копирования уже существующей точки
- Конструктор по умолчанию с координатами (0, 0)

Для доступа к полям предусмотрены методы геттеры и сеттеры, обеспечивающие контролируемый доступ к состоянию объекта.



```
package functions;

public class FunctionPoint {  18 usages  new *
    private double x;  4 usages
    private double y;  4 usages

    // геттеры точек
    public double getX(){  19 usages  new *
        return x;
    }
    public double getY(){  4 usages  new *
        return y;
    }

    // сетеры значений точек
    public void setX(double x){  1 usage  new *
        this.x=x;
    }
    public void setY(double y){  1 usage  new *
         this.y=y;
    }


    public FunctionPoint(double x, double y){ // конструктор с заданными координатами  5 usages  new *
        this.x = x;
        this.y = y;
    }
    public FunctionPoint(FunctionPoint point){ // конструктор копирования  3 usages  new *
        this(point.x, point.y);
    }
    public FunctionPoint(){ // конструктор по умолчанию (точка 0,0)  no usages  new *
        this( x: 0.0,  y: 0.0);
    }
}
```

Рисунок 1

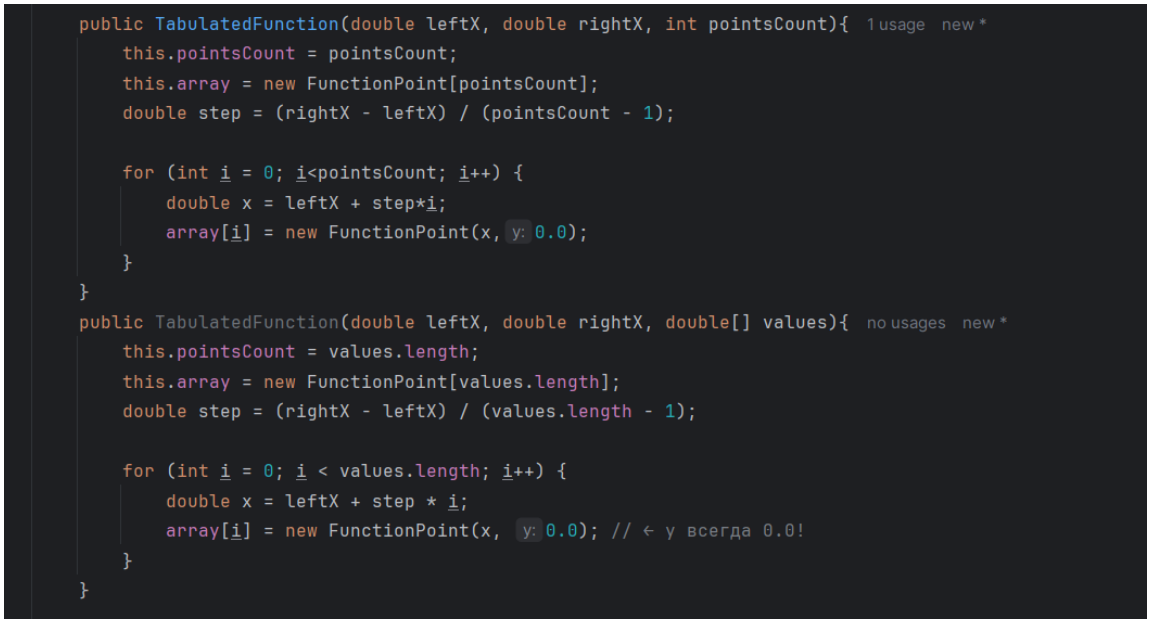
Задание №3

Для выполнения задания 3 я написала основной класс «TabulatedFunction» для представления табулированной функции. В качестве хранилища данных используется массив объектов «FunctionPoint», при этом обеспечивается поддержание порядка точек по возрастанию координаты x.

Реализованные конструкторы класса «TabulatedFunction»:

- Создание функции с заданным количеством точек (значения y инициализируются нулями)
- Создание функции с количеством точек, определяемым длиной массива values (значения y(игрек) также нулевые)

В обоих случаях точки распределяются равномерно по заданному интервалу [leftX, rightX].



```
public TabulatedFunction(double leftX, double rightX, int pointsCount){  1 usage  new *
    this.pointsCount = pointsCount;
    this.array = new FunctionPoint[pointsCount];
    double step = (rightX - leftX) / (pointsCount - 1);

    for (int i = 0; i<pointsCount; i++) {
        double x = leftX + step*i;
        array[i] = new FunctionPoint(x, y: 0.0);
    }
}
public TabulatedFunction(double leftX, double rightX, double[] values){  no usages  new *
    this.pointsCount = values.length;
    this.array = new FunctionPoint[values.length];
    double step = (rightX - leftX) / (values.length - 1);

    for (int i = 0; i < values.length; i++) {
        double x = leftX + step * i;
        array[i] = new FunctionPoint(x,  y: 0.0); // ← y всегда 0.0!
    }
}
```

Рисунок 2

Задание №4

Для выполнения задания 4 мной были описаны методы, обеспечивающие начальный функционал работы с табулированной функцией:

- «getLeftDomainBorder()» который возвращает x-координату начальной точки
- «getRightDomainBorder()» который возвращает x-координату конечной точки
- «getFunctionValue(x)» который вычисляет значение функции в произвольной точке с использованием линейной интерполяции (также с использованием машинного эпсилона для точности)

Для точек внутри области определения выполняется расчет значения по уравнению прямой, проходящей через две соседние точки таблицы. Для точек вне области определения возвращается специальное значение NaN(тип double).

```
// методы
public double getLeftDomainBorder(){  no usages  new *
    return array[0].getX();
}
public double getRightDomainBorder(){  no usages  new *
    return array[pointsCount-1].getX();
}
public double getFunctionValue(double x) {  5 usages  new *
    if (x< array[0].getX() || x > array[pointsCount - 1].getX()) {
        return Double.NaN;
    }

    for (int i = 0; i< pointsCount - 1; i++) {
        double x1 = array[i].getX();
        double x2 = array[i + 1].getX();

        if (x >= x1 && x <= x2) {
            if (Math.abs(x - x1) < 1e-10) {
                return array[i].getY();
            }
            if (Math.abs(x - x2) < 1e-10) {
                return array[i + 1].getY();
            }

            double slope = (array[i + 1].getY() - array[i].getY()) /
                    (array[i + 1].getX() - array[i].getX());
            return array[i].getY() + slope * (x - array[i].getX());
        }
    }
    return Double.NaN;
}
```

Рисунок 3

Задание №5

Задание 5 требует реализации комплекса методов для управления отдельными точками табулированной функции:

- «getPointsCount()» для получения количества точек
- «getPoint(index)» для получения копии точки (с соблюдением инкапсуляции)
- «setPoint(index, point)»для замены точки с проверкой сохранения порядка
- Методы доступа и модификации координат x и y

Все методы изменения координат x включают проверки, предотвращающие нарушение упорядоченности точек.

```
public int getPointsCount(){  5 usages  new *
    return pointsCount;
}

public FunctionPoint getPoint(int index){  no usages  new *
    return new FunctionPoint(array[index]);
}
public void setPoint(int index, FunctionPoint point) {  no usages  new *
    double newX = point.getX();
    if (index>0 && newX <=array[index-1].getX()) return;
    if (index<pointsCount-1 && newX>=array[index+1].getX()) return;

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

public void setPointX(int index, double x) {...}
public void setPointY(int index, double y) {  3 usages  new *
    array[index].setY(y);
}
public double getPointX(int index) { return array[index].getX(); }
public double getPointY(int index) { return array[index].getY(); }
```

Рисунок 4

Задание №6

Выполнение задания 6 требовало реализации методов динамического изменения структуры табулированной функции:

- «deletePoint(index)» для удаления точки со сдвигом последующих элементов
- «addPoint(point)» для вставки новой точки с сохранением порядка

Методы используют алгоритмы работы с массивом, включая «System.arraycopy()» для быстрого копирования данных и ручной сдвиг элементов, что минимизирует создание временных массивов.

```
    public void deletePoint(int index){  1 usage  new *

        System.arraycopy(array,  srcPos: index+1, array, index,  length: pointsCount-index-1);
        pointsCount--;
        array[pointsCount] = null;
    }

    public void addPoint(FunctionPoint point){  1 usage  new *
        FunctionPoint newPoint = new FunctionPoint(point);
        int newIndex = 0;
        while (newIndex<pointsCount && array[newIndex].getX()< newPoint.getX()){
            newIndex++;
        }
        if (newIndex< pointsCount && array[newIndex].getX() == newPoint.getX()){
            array[newIndex] = newPoint;
            return;
        }
        if (pointsCount >= array.length) {
            FunctionPoint[] newArray = new FunctionPoint[array.length * 2];
            System.arraycopy(array,  srcPos: 0, newArray,  destPos: 0, pointsCount);
            array = newArray;
        }
        for (int i = pointsCount; i>newIndex; i--) {
            array[i] = array[i - 1];
        }
        array[newIndex] = newPoint;
        pointsCount++;
    }
}
```

Рисунок 5

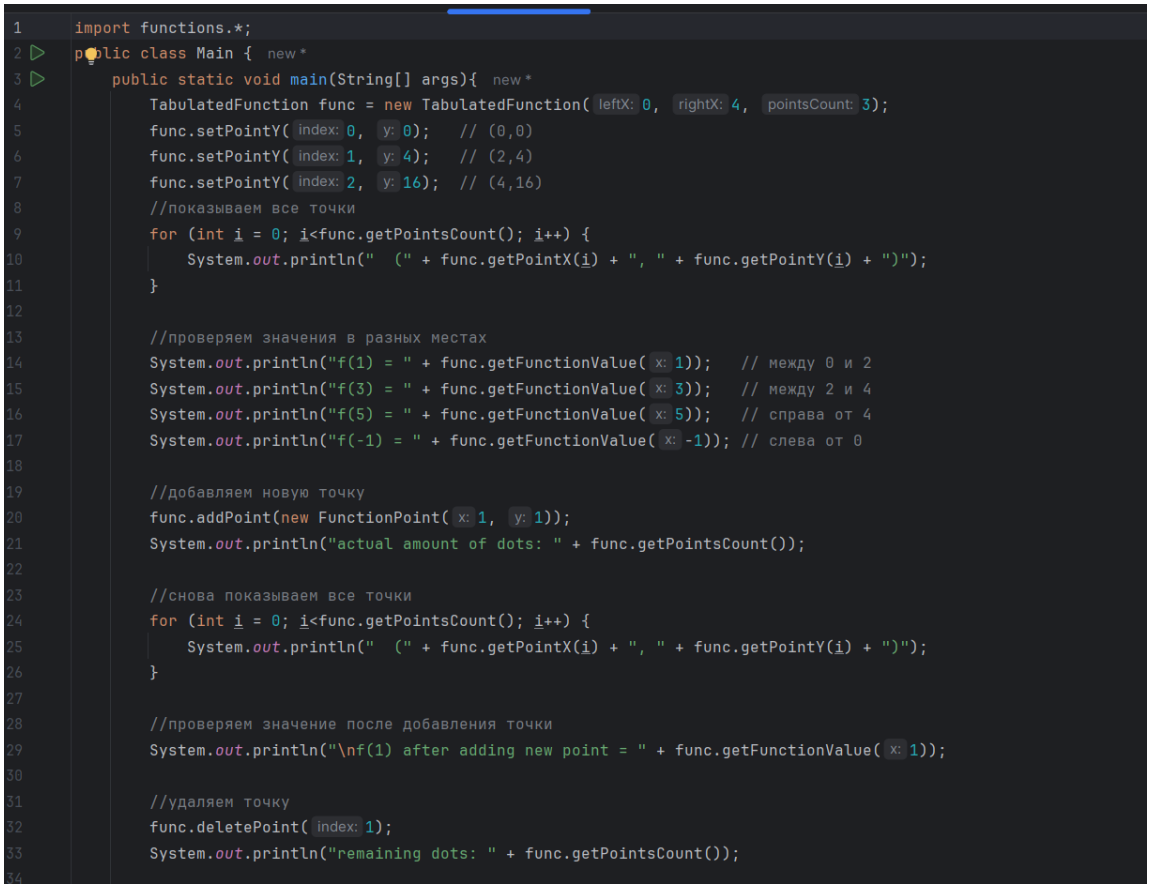
Задание №7

Мной был создан тестовый класс «Main» с демонстрацией работы всех реализованных возможностей:

Необходимые проверки:

- Создание функции с тремя точками на интервале [0, 4]
- Вывод исходного состояния точек
- Расчет значений функции в различных точках (внутри и вне области определения)
- Добавление новой точки с проверкой сортировки
- Удаление точки и проверка целостности данных
- Поэтапный вывод изменений структуры функции

По результатам тестирования методы работают корректно, линейная интерполяция вычисляет значения с ожидаемой точностью, операции добавления и удаления точек сохраняют упорядоченность данных.



```
import functions.*;
public class Main {
    public static void main(String[] args){
        TabulatedFunction func = new TabulatedFunction( leftX: 0,  rightX: 4,  pointsCount: 3);
        func.setPointY( index: 0,  y: 0);   // (0,0)
        func.setPointY( index: 1,  y: 4);   // (2,4)
        func.setPointY( index: 2,  y: 16);  // (4,16)
        //показываем все точки
        for (int i = 0; i<func.getPointsCount(); i++) {
            System.out.println("  (" + func.getPointX(i) + ", " + func.getPointY(i) + ")");
        }

        //проверяем значения в разных местах
        System.out.println("f(1) = " + func.getFunctionValue( x: 1));   // между 0 и 2
        System.out.println("f(3) = " + func.getFunctionValue( x: 3));   // между 2 и 4
        System.out.println("f(5) = " + func.getFunctionValue( x: 5));   // справа от 4
        System.out.println("f(-1) = " + func.getFunctionValue( x: -1)); // слева от 0

        //добавляем новую точку
        func.addPoint(new FunctionPoint( x: 1,  y: 1));
        System.out.println("actual amount of dots: " + func.getPointsCount());

        //снова показываем все точки
        for (int i = 0; i<func.getPointsCount(); i++) {
            System.out.println("  (" + func.getPointX(i) + ", " + func.getPointY(i) + ")");
        }

        //проверяем значение после добавления точки
        System.out.println("\nf(1) after adding new point = " + func.getFunctionValue( x: 1));

        //удаляем точку
        func.deletePoint( index: 1);
        System.out.println("remaining dots: " + func.getPointsCount());
```

Рисунок 6

```
        //финальные точки
        for (int i = 0; i<func.getPointsCount(); i++) {
            System.out.println("  (" + func.getPointX(i) + ", " + func.getPointY(i) + ")");
        }
        //вроде все проверила
    }
}
```

Рисунок 7

```
Main
C:\Users\sweet\.jdks\openjdk-24.0.2+12-54\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.2\lib\idea_rt.jar=3293" -Dfile.encoding=UTF-8
  (0.0, 0.0)
  (2.0, 4.0)
  (4.0, 16.0)
f(1) = 2.0
f(3) = 10.0
f(5) = NaN
f(-1) = NaN
actual amount of dots: 4
  (0.0, 0.0)
  (1.0, 1.0)
  (2.0, 4.0)
  (4.0, 16.0)

f(1) after adding new point = 1.0
remaining dots: 3
  (0.0, 0.0)
  (2.0, 4.0)
  (4.0, 16.0)

Process finished with exit code 0
```

Рисунок 8