

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет имени
академика С.П. Королева»
Институт информатики и кибернетики Кафедра
технической кибернетики

Отчет по лабораторной работе №2

Дисциплина: «ООП»

Тема «из названия лабораторной работы»

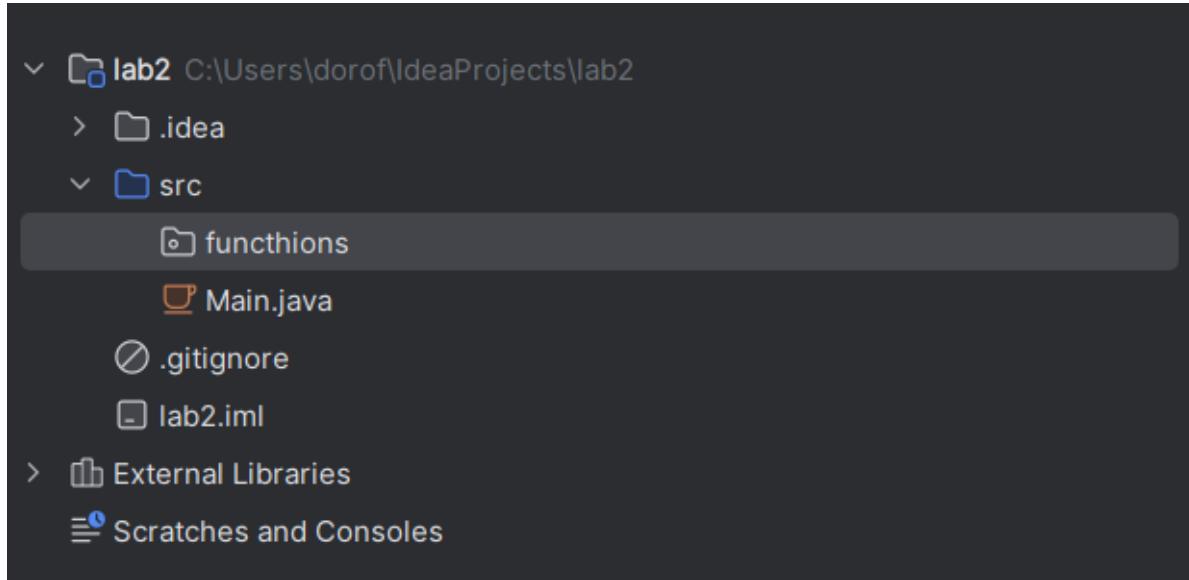
Выполнил: Дорофеева Александра Алексеевна

Группа: 6201-120303D

Проверил: преподаватель Борисов Д.С.

Самара, 2025

Задание 1



Создаем пакет functions, в котором далее будут создаваться классы программы.

Задание 2

The screenshot shows the IntelliJ IDEA code editor with two tabs: 'Main.java' and 'FunctionPoint.java'. The 'FunctionPoint.java' tab is active, showing the following code:

```
1 package functhions;
2
3 public class FunctionPoint { no usages
4 }
5 |
```

В пакете functions создаем класс FunctionPoint

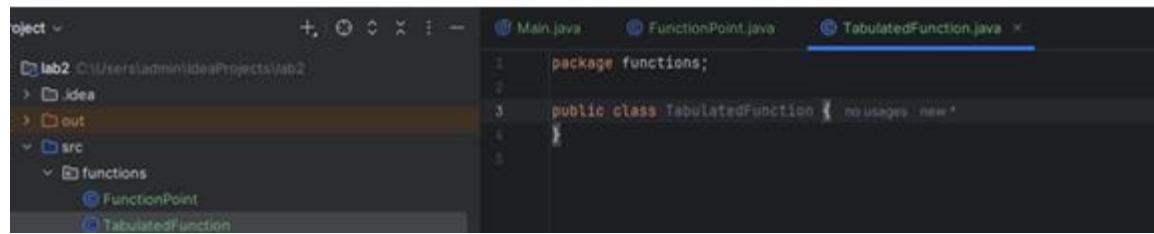
Main.java × FunctionPoint.java ×

```
1 package functions;
2
3 public class FunctionPoint { 1 usage
4     private double x; 5 usages
5     private double y; 5 usages
6
7     // Конструктор с заданными координатами
8     public FunctionPoint(double x, double y) { 1 usage
9         this.x = x;
10        this.y = y;
11    }
12
13     // Конструктор копирования
14     public FunctionPoint(FunctionPoint point) { no usages
15         this.x = point.x;
16         this.y = point.y;
17     }
18
19     // Конструктор по умолчанию
20     public FunctionPoint() { no usages
21         this(x: 0.0, y: 0.0);
22     }
23
24     // Геттеры и сеттеры
25     public double getX() { no usages
26         return x;
27     }
28
29     public void setX(double x) { no usages
30         this.x = x;
31     }
32
33     public double getY() { no usages
34         return y;
35     }
36
37     public void setY(double y) { no usages
38         this.y = y;
39     }
40 }
```

Реализован класс FunctionPoint, инкапсулирующий координаты точки (x,y) через приватные поля с открытыми геттерами/сеттерами и тремя конструкторами: с

параметрами, копирования и по умолчанию, обеспечивающий безопасную работу с точками функции.

Задание 3



The screenshot shows the IntelliJ IDEA interface. The left sidebar displays the project structure with a tree view of files and folders. In the center, there is a code editor window with three tabs at the top: Main.java, FunctionPoint.java, and TabulatedFunction.java. The TabulatedFunction.java tab is currently active. The code in the editor is:

```
package functions;
public class TabulatedFunction { }
```

В пакете functions создаем класс TabulatedFunction

```
1 package functions;
2
3 public class TabulatedFunction { 5 usages  ↗ Dorofeeva Aleksandra *
4     private FunctionPoint[] points; 33 usages
5     private int pointsCount; 17 usages
6     private static final int INITIAL_CAPACITY = 10; 2 usages
7     private static final double EPSILON = 1e-9; // Машинный эпсилон 11 usages
8
9     // ЗАДАНИЕ 3: Конструкторы
10    public TabulatedFunction(double leftX, double rightX, int pointsCount) { 1 usage  ↗ Doro
11        if (pointsCount < 2) {
12            pointsCount = 2;
13        }
14
15        this.pointsCount = pointsCount;
16        this.points = new FunctionPoint[Math.max(pointsCount * 2, INITIAL_CAPACITY)];
17
18        double step = (rightX - leftX) / (pointsCount - 1);
19        for (int i = 0; i < pointsCount; i++) {
20            double x = leftX + i * step;
21            points[i] = new FunctionPoint(x, y: 0.0);
22        }
23    }
24
25 @
26    public TabulatedFunction(double leftX, double rightX, double[] values) { 1 usage  ↗ Doro
27        this.pointsCount = values.length;
28        this.points = new FunctionPoint[Math.max(values.length * 2, INITIAL_CAPACITY)];
29
30        double step = (rightX - leftX) / (values.length - 1);
31        for (int i = 0; i < values.length; i++) {
32            double x = leftX + i * step;
33            points[i] = new FunctionPoint(x, values[i]);
34        }
35    }
36 }
```

Создан класс TabulatedFunction с двумя конструкторами, формирующими массив точек FunctionPoint с равномерным распределением по X: первый создает pointsCount точек с Y=0, второй использует заданный массив значений Y, автоматически рассчитывая шаг между точками.

Задание 4

```
// ЗАДАНИЕ 4: Методы области определения и вычисления
public double getLeftDomainBorder() { 2 usages  ↗ Dorofeeva Alexsandra
    return points[0].getX();
}

public double getRightDomainBorder() { 2 usages  ↗ Dorofeeva Alexsandra
    return points[pointsCount - 1].getX();
}

public double getFunctionValue(double x) { 3 usages  ↗ Dorofeeva Alexsandra *
    if (x < getLeftDomainBorder() - EPSILON || x > getRightDomainBorder() + EPSILON) {
        return Double.NaN;
    }

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

        // Проверяем, находится ли x между x1 и x2
        if (x >= x1 - EPSILON && x <= x2 + EPSILON) {
            if (Math.abs(x1 - x2) < EPSILON) {
                return points[i].getY(); // Если точки совпадают, возвращаем у
            }

            double y1 = points[i].getY();
            double y2 = points[i + 1].getY();

            // Если x равно x1 или x2, возвращаем соответствующее значение у
            if (Math.abs(x - x1) < EPSILON) {
                return y1;
            }
            if (Math.abs(x - x2) < EPSILON) {
                return y2;
            }

            // Линейная интерполяция между y1 и y2
            return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
        }
    }
}
```

```
return Double.NaN; // Если x не попадает ни в один интервал
```

В методе setUp инициализируется массив точек и экземпляр класса Function для работы с ними. Тестирование включает проверку корректности возвращаемых значений методов getLeftDomainBorder и getRightDomainBorder для границ области определения, верификацию вычисления функции в известных точках

(включая граничные), оценку правильности линейной интерполяции между узлами и проверку возврата Double.NaN при выходе за пределы области определения.

Задание 5

```
// ЗАДАНИЕ 5: Методы работы с точками
public int getPointsCount() { 4 usages  ↗ Dorofeeva Alexsandra
    return pointsCount;
}

public FunctionPoint getPoint(int index) {  no usages  ↗ Dorofeeva Alexsandra
    return new FunctionPoint(points[index]);
}

public void setPoint(int index, FunctionPoint point) {  1 usage  ↗ Dorofeeva Alexsandra *
    if (index > 0 && point.getX() <= points[index - 1].getX() + EPSILON) {
        return;
    }
    if (index < pointsCount - 1 && point.getX() >= points[index + 1].getX() - EPSILON) {
        return;
    }

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

public double getPointX(int index) {  5 usages  ↗ Dorofeeva Alexsandra
    return points[index].getX();
}

public void setPointX(int index, double x) {  no usages  ↗ Dorofeeva Alexsandra *
    if (index > 0 && x <= points[index - 1].getX() + EPSILON) {
        return;
    }
    if (index < pointsCount - 1 && x >= points[index + 1].getX() - EPSILON) {
        return;
    }

    points[index].setX(x);
}

public double getPointY(int index) {  4 usages  ↗ Dorofeeva Alexsandra
    return points[index].getY();
}

public void setPointY(int index, double v) {  1 usage  ↗ Dorofeeva Alexsandra
```

```

    public double getPointY(int index) { 4 usages  ↗ Dorofeeva Alexsandra
        return points[index].getY();
    }

    public void setPointY(int index, double y) { 1 usage  ↗ Dorofeeva Alexsandra
        points[index].setY(y);
    }
}

```

Реализованы методы доступа к точкам: getPointsCount(), getPoint() (возвращает копию), setPoint() с проверкой порядка, геттеры/сеттеры для координат X и Y, обеспечивающие инкапсуляцию и сохранение упорядоченности точек.

Задание 6

```

// ЗАДАНИЕ 6: Методы изменения количества точек
public void deletePoint(int index) { 1 usage  ↗ Dorofeeva Alexsandra
    if (pointsCount <= 2) {
        return;
    }

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

public void addPoint(FunctionPoint point) { 2 usages  ↗ Dorofeeva Alexsandra *
    int insertIndex = 0;
    while (insertIndex < pointsCount && points[insertIndex].getX() < point.getX() - EPSILON) {
        insertIndex++;
    }

    if (insertIndex < pointsCount && Math.abs(points[insertIndex].getX() - point.getX()) < EPSILON) {
        return;
    }

    if (pointsCount == points.length) {
        FunctionPoint[] newPoints = new FunctionPoint[points.length * 2];
        System.arraycopy(points, [srcPos: 0], newPoints, [destPos: 0], pointsCount);
        points = newPoints;
    }

    System.arraycopy(points, insertIndex, points, [destPos: insertIndex + 1], [length: pointsCount - insertIndex]);

    points[insertIndex] = new FunctionPoint(point);
    pointsCount++;
}
}

```

Добавлены методы deletePoint() для удаления точки (сохраняя минимум 2 точки) и addPoint() для вставки новой точки с автоматическим расширением массива и сохранением порядка по X через System.arraycopy().

Задание 7

```
>Main.java × FunctionPoint.java TabulatedFunction.java

1 import functions.FunctionPoint;
2 import functions.TabulatedFunction;
3
4 public class Main {
5     public static void main(String[] args) {
6         double left = 0;
7         double right = 4;
8         int count = 5;
9
10        // Создание функции
11        TabulatedFunction func = new TabulatedFunction(left, right, count);
12
13        System.out.println("Функция x^3");
14        System.out.println("Границы функции:");
15        System.out.println("Левая граница: " + func.getLeftDomainBorder());
16        System.out.println("Правая граница: " + func.getRightDomainBorder());
17
18        // Задание значений y = x^3
19        for (int i = 0; i < func.getPointsCount(); i++) {
20            double x = func.getPointX(i);
21            func.setPointY(i, y: x * x * x);
22        }
23
24        // Проверка значений функции
25        System.out.println("Проверка значений функции:");
26        for (double z = -1; z <= 5; z += 1.0) {
27            System.out.println("f(" + z + ") = " + func.getFunctionValue(z));
28        }
29
30        // Создание функции с готовыми значениями
31        double[] values = {0, 1, 8, 27, 64};
32        TabulatedFunction func1 = new TabulatedFunction(left, right, values);
33
34        System.out.println("Функция создана с массивом значений:");
35        for (int i = 0; i < func1.getPointsCount(); i++) {
36            System.out.println("x=" + func1.getPointX(i) + ", y=" + func1.getPointY(i));
37        }
38
39        // Изменение точки
40        System.out.println("Изменяем значение у второй точки:");
41        func.setPoint(index: 1, new FunctionPoint(x: 1.5, y: 3.375));
```

```
42     System.out.println("Новая точка 1: x = " + func.getPointX(index: 1) + ", y = " + func.getPointY(index: 1));
43
44     // Добавление точки
45     System.out.println("Добавляем новую точку (x=2.5, y=15.625):");
46     func.addPoint(new FunctionPoint(x: 2.5, y: 15.625));
47     for (int i = 0; i < func.getPointsCount(); i++) {
48         System.out.println(" (" + func.getPointX(i) + "; " + func.getPointY(i) + ")");
49     }
50
51     // Интерполяция и добавление новой точки
52     System.out.println("Находим значение Y для 3.5 с помощью линейной интерполяции:");
53     func.addPoint(new FunctionPoint(x: 3.5, func.getFunctionValue(x: 3.5)));
54     System.out.println("Добавлена новая точка (3.5; " + func.getFunctionValue(x: 3.5) + ")");
55
56     // Удаление точки
57     System.out.println("Удаляем третью точку:");
58     func.deletePoint(index: 2);
59     for (int i = 0; i < func.getPointsCount(); i++) {
60         System.out.println(" (" + func.getPointX(i) + "; " + func.getPointY(i) + ")");
61     }
62 }
63 }
```

Протестирована работа классов через Main: создана функция $f(x)=x^2$, проверены вычисления значений (включая интерполяцию), добавление/удаление точек и обработка граничных случаев, подтвердившие корректность реализации.

Проверяем

```
"C:\Program Files\Java\jdk-25\bin\java.exe" "-javaagent:G:\intellij IDEA\lib\idea_rt.jar" -Dfile.encoding=UTF-8
Функция x^3
Границы функции:
Левая граница: 0.0
Правая граница: 4.0
Проверка значений функции:
f(-1.0) = NaN
f(0.0) = 0.0
f(1.0) = 1.0
f(2.0) = 8.0
f(3.0) = 27.0
f(4.0) = 64.0
f(5.0) = NaN
Функция создана с массивом значений:
x=0.0, y=0.0
x=1.0, y=1.0
x=2.0, y=8.0
x=3.0, y=27.0
x=4.0, y=64.0
Изменяем значение у второй точки:
Новая точка 1: x = 1.5, y = 3.375
Добавляем новую точку (x=2.5, y=15.625):
(0.0; 0.0)
(1.5; 3.375)
(2.0; 8.0)
(2.5; 15.625)
(3.0; 27.0)
(4.0; 64.0)
Находим значение Y для 3.5 с помощью линейной интерполяции:
Добавлена новая точка (3.5; 45.5)
Удаляем третью точку:
(0.0; 0.0)
(1.5; 3.375)
(2.5; 15.625)
(3.0; 27.0)
(3.5; 45.5)
(4.0; 64.0)
```