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»

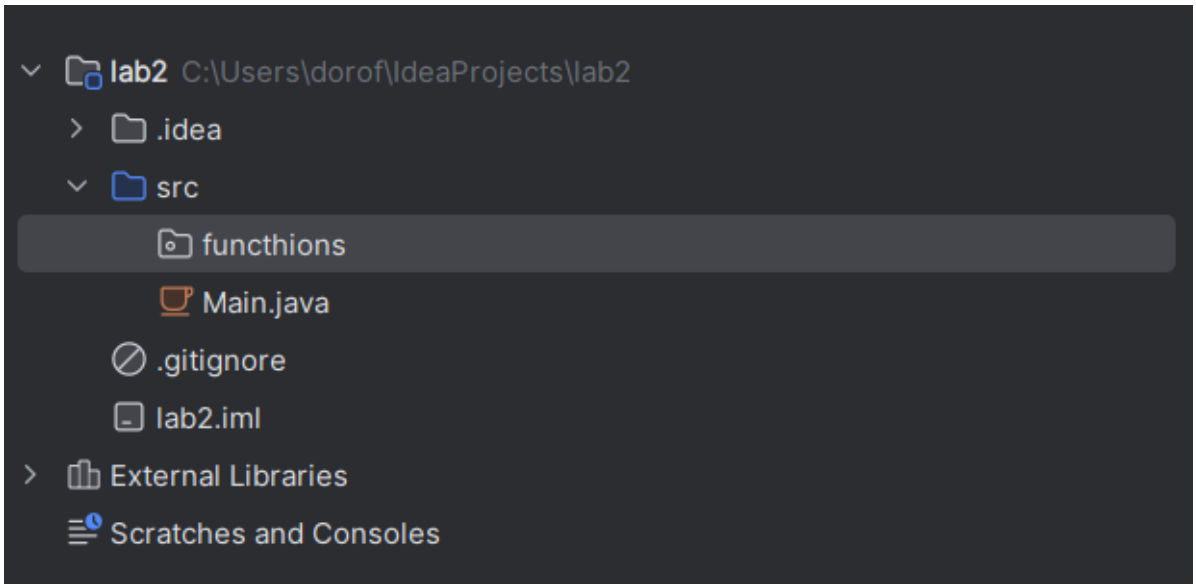
Выполнил: Дорофеева Александра Алексеевна

Группа: 6201-120303D

Проверил: преподаватель Борисов Д.С.

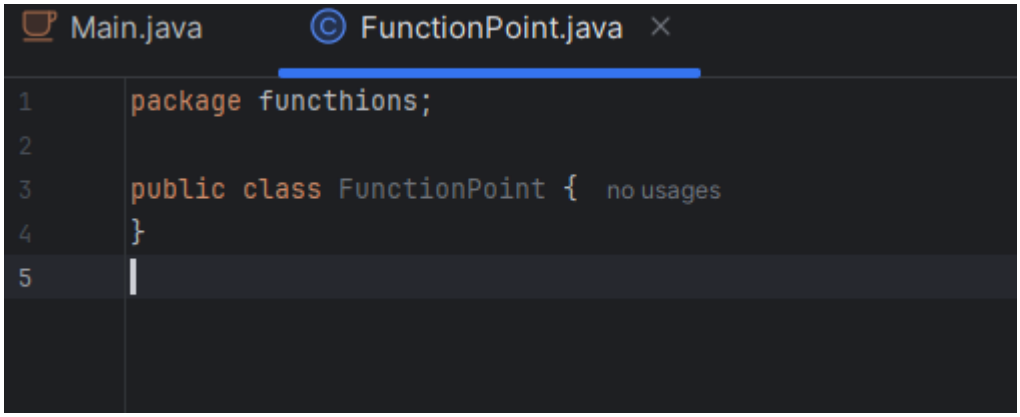
Самара, 2025

Задание 1



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

Задание 2



В пакете functions создаем класс FunctionPoint

Main.java × FunctionPoint.java ×

```java
package functions;

public class FunctionPoint {  1 usage
    private double x;  5 usages
    private double y;  5 usages

    // Конструктор с заданными координатами
    public FunctionPoint(double x, double y) {  1 usage
        this.x = x;
        this.y = y;
    }

    // Конструктор копирования
    public FunctionPoint(FunctionPoint point) {  no usages
        this.x = point.x;
        this.y = point.y;
    }

    // Конструктор по умолчанию
    public FunctionPoint() {  no usages
        this( x: 0.0,  y: 0.0);
    }

    // Геттеры и сеттеры
    public double getX() {  no usages
        return x;
    }

    public void setX(double x) {  no usages
        this.x = x;
    }

    public double getY() {  no usages
        return y;
    }

    public void setY(double y) {  no usages
        this.y = y;
    }
}
```

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

В пакете functions создаем класс TabulatedFunction

```java
package functions;

public class TabulatedFunction {  no usages
    private FunctionPoint[] points;  33 usages
    private int pointsCount;  17 usages
    private static final int INITIAL_CAPACITY = 10;  2 usages

    // ЗАДАНИЕ 3: Конструкторы
    public TabulatedFunction(double leftX, double rightX, int pointsCount) {  no usages
        if (pointsCount < 2) {
            pointsCount = 2;
        }

        this.pointsCount = pointsCount;
        this.points = new FunctionPoint[Math.max(pointsCount * 2, INITIAL_CAPACITY)];

        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i < pointsCount; i++) {
            double x = leftX + i * step;
            points[i] = new FunctionPoint(x,  y: 0.0);
        }
    }

    public TabulatedFunction(double leftX, double rightX, double[] values) {  no usages
        this.pointsCount = values.length;
        this.points = new FunctionPoint[Math.max(values.length * 2, INITIAL_CAPACITY)];

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

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

```
36      public double getLeftDomainBorder() {  1 usage
37          return points[0].getX();
38      }
39
40      public double getRightDomainBorder() {  1 usage
41          return points[pointsCount - 1].getX();
42      }
43
44      public double getFunctionValue(double x) {  no usages
45          if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
46              return Double.NaN;
47          }
48
49          for (int i = 0; i < pointsCount - 1; i++) {
50              double x1 = points[i].getX();
51              double x2 = points[i + 1].getX();
52
53              if (x >= x1 && x <= x2) {
54                  if (x1 == x2) {
55                      return points[i].getY();
56                  }
57
58                  double y1 = points[i].getY();
59                  double y2 = points[i + 1].getY();
60                  return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
61              }
62          }
63
64          return Double.NaN;
65      }
66
```

Добавлены методы getLeftDomainBorder() и getRightDomainBorder() для получения границ области определения, а также getFunctionValue() с линейной интерполяцией между соседними точками, возвращающий Double.NaN для значений вне области определения.

Задание 5

```java
    // ЗАДАНИЕ 5: Методы работы с точками
    public int getPointsCount() {  no usages
        return pointsCount;
    }

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

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

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

    public double getPointX(int index) {  no usages
        return points[index].getX();
    }

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

        points[index].setX(x);
    }

    public double getPointY(int index) {  no usages
        return points[index].getY();
    }
```

```java
    public double getPointY(int index) {  no usages
        return points[index].getY();
    }

    public void setPointY(int index, double y) {  no usages
        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

```java
    // ЗАДАНИЕ 6: Методы изменения количества точек
    public void deletePoint(int index) {  no usages
        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) {  no usages
        int insertIndex = 0;
        while (insertIndex < pointsCount && points[insertIndex].getX() < point.getX()) {
            insertIndex++;
        }

        if (insertIndex < pointsCount && points[insertIndex].getX() == point.getX()) {
            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

```java
import functions.FunctionPoint;
import functions.TabulatedFunction;

public class Main {
    public static void main(String[] args) {
        double left = 0;
        double right = 4;
        int count = 5;

        // Создание функции
        TabulatedFunction func = new TabulatedFunction(left, right, count);

        System.out.println("Функция x^3");
        System.out.println("Границы функции:");
        System.out.println("Левая граница: " + func.getLeftDomainBorder());
        System.out.println("Правая граница: " + func.getRightDomainBorder());

        // Задание значений y = x^3
        for (int i = 0; i < func.getPointsCount(); i++) {
            double x = func.getPointX(i);
            func.setPointY(i, y: x * x * x);
        }

        // Проверка значений функции
        System.out.println("Проверка значений функции:");
        for (double z = -1; z <= 5; z += 1.0) {
            System.out.println("f(" + z + ") = " + func.getFunctionValue(z));
        }

        // Создание функции с готовыми значениями
        double[] values = {0, 1, 8, 27, 64};
        TabulatedFunction func1 = new TabulatedFunction(left, right, values);

        System.out.println("Функция создана с массивом значений:");
        for (int i = 0; i < func1.getPointsCount(); i++) {
            System.out.println("x=" + func1.getPointX(i) + ", y=" + func1.getPointY(i));
        }

        // Изменение точки
        System.out.println("Изменяем значение y второй точки:");
        func.setPoint( index: 1, new FunctionPoint( x: 1.5, y: 3.375));
```

```
        System.out.println("Новая точка 1: x = " + func.getPointX( index: 1) + ", y = " + func.getPointY( index: 1));

        // Добавление точки
        System.out.println("Добавляем новую точку (x=2.5, y=15.625):");
        func.addPoint(new FunctionPoint( x: 2.5,  y: 15.625));
        for (int i = 0; i < func.getPointsCount(); i++) {
            System.out.println("(" + func.getPointX(i) + "; " + func.getPointY(i) + ")");
        }

        // Интерполяция и добавление новой точки
        System.out.println("Находим значение Y для 3.5 с помощью линейной интерполяции:");
        func.addPoint(new FunctionPoint( x: 3.5, func.getFunctionValue( x: 3.5)));
        System.out.println("Добавлена новая точка (3.5; " + func.getFunctionValue( x: 3.5) + ")");

        // Удаление точки
        System.out.println("Удаляем третью точку:");
        func.deletePoint( index: 2);
        for (int i = 0; i < func.getPointsCount(); i++) {
            System.out.println("(" + func.getPointX(i) + "; " + func.getPointY(i) + ")");
        }
    }
}
```

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:\i
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 y 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)

Process finished with exit code 0
```