Лабораторная работа №2

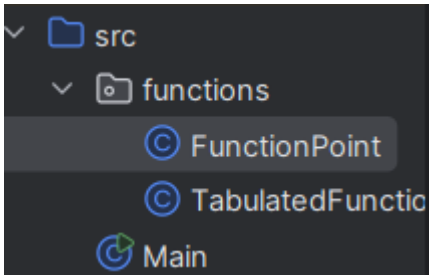
Лебедев Кирилл Дмитриевич 6204-010302D

Задание 1

Создать пакет functions

Ход выполнения:

- Создан пакет functions в структуре проекта
- Все последующие классы размещены в данном пакете



Результат: Пакет создан успешно.

Задание 2

Создать класс FunctionPoint

Ход выполнения:

- Реализован класс с приватными полями x и y
- Созданы конструкторы:
  - FunctionPoint(double x, double y)
  - FunctionPoint(FunctionPoint point) (копирующий)
  - FunctionPoint() (по умолчанию)
- Реализованы геттеры для доступа к координатам

```java
// Создаёт объект точки с координатами
public FunctionPoint(double x, double y) {  3 usages
    this.x = x;
    this.y = y;
}

// Создаёт объект точки с теми же координатами точки
public FunctionPoint(FunctionPoint point) {  2 usages
    this.x = point.x;
    this.y = point.y;
}

// Создаёт объект точки с координатами (0;0)
public FunctionPoint() {  no usages
    this.x = 0;
    this.y = 0;
}
```

```java
// Задать X
public void setX(double x) { this.x = x; }

// Получить X
public double getX() {  16 usages
    return x;
}

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

// Получить Y
public double getY() {  8 usages
    return y;
}
```

Результат: Класс создан, инкапсуляция обеспечена.

Задание 3

Создать класс TabulatedFunction

Ход выполнения:

- Реализован класс с массивом FunctionPoint[] points
- Добавлено поле pointsCount для отслеживания количества точек
- Реализованы конструкторы:
    - TabulatedFunction(double leftX, double rightX, int pointsCount)
    - TabulatedFunction(double leftX, double rightX, double[] values)
- Точки создаются через равные интервалы

```
// Конструктор создания объекта табулированной функции через кол-во точек
public TabulatedFunction(double leftX, double rightX, int pointsCount) {  no usages
    this.pointsCount = pointsCount;
    this.points = new FunctionPoint[pointsCount*2];

    double step = (rightX - leftX) / (pointsCount - 1); // Задание шага

    for (int i = 0; i < pointsCount; i++) {
        double x = leftX + step * i;
        points[i] = new FunctionPoint(x,  y: 0);    // создание точек с равным шагом step через интервал x
    }
}

// Конструктор создания объекта табулированной функции через массив значений
public TabulatedFunction(double leftX, double rightX, double[] values) {  1 usage
    this.pointsCount = values.length;
    this.points = new FunctionPoint[values.length*2];

    double step = (rightX - leftX) / (values.length - 1);   // Задание шага

    for (int i = 0; i < values.length; i++) {
        double x = leftX + step * i;
        points[i] = new FunctionPoint(x, values[i]);    // создание точек с равным шагом step через интервал x
    }
}
```

Результат: Конструкторы работают корректно, точки упорядочены по x.

Задание 4

Методы области определения и вычисления значения

Ход выполнения:

- Реализованы методы границ области определения
- Реализован метод getFunctionValue() с линейной интерполяцией
- Добавлена проверка на выход за границы области определения

```java
public double getLeftDomainBorder() { return points[0].getX(); }

public double getRightDomainBorder() { return points[pointsCount - 1].getX(); }

// интерпол
public double getFunctionValue(double x) {  5 usages
    if (x < getLeftDomainBorder() || x > getRightDomainBorder()) { return Double.NaN; } // Вне графика
    if (x == getLeftDomainBorder()) { return points[0].getY(); }    // на левом конце
    if (x == getRightDomainBorder()) { return points[pointsCount - 1].getY(); }     // на правом конце

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

        if (x >= x1 && x <= x2) {
            double y1 = points[i].getY();
            double y2 = points[i + 1].getY();

            return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
        }
    }
    return Double.NaN;
}
```

Результат: Методы работают корректно, интерполяция выполняется.

Задание 5

Методы работы с точками

Ход выполнения:

- Реализованы методы доступа к точкам
- Добавлены проверки в методах изменения координат
- Обеспечена инкапсуляция через возвращение копий

```java
// кол-во точек
public int getPointsCount() { return pointsCount; }

// копия точки
public FunctionPoint getPoint(int index) {  5 usages
    return new FunctionPoint(points[index]);
}
```

Результат: Все методы работают, инкапсуляция соблюдена.

Задание 6

Изменение количества точек

Ход выполнения:

- Реализовано динамическое изменение массива

- Добавлены методы deletePoint() и addPoint()
- Обеспечено сохранение порядка точек

```
public void deletePoint(int index) {  1 usage
    if (pointsCount <= 2) { return; }   // если точек всего 2 - выйти
    System.arraycopy(points,  srcPos: index+1,points, index,  length: pointsCount-index-1);
    pointsCount--;
}

// добавить новую точку табулированной функции
public void addPoint(FunctionPoint point) {  1 usage
    FunctionPoint[] newPoints = new FunctionPoint[pointsCount*2];

    // ищем позицию между точками
    int pos = 0;
    while (pos < pointsCount && points[pos].getX() < point.getX()) {
        pos++;
    }

    // копируем до нужной позиции
    System.arraycopy(points,  srcPos: 0, newPoints,  destPos: 0, pos);

    // вставляем точку
    newPoints[pos] = point;

    // копируем остаток
    System.arraycopy(points, pos, newPoints,  destPos: pos+1,  length: pointsCount - pos);
    // меняем обратно и увеличиваем кол-во точек
    points = newPoints;
    pointsCount++;
}
```

Результат: Методы работают эффективно, порядок точек сохраняется.

Задание 7

Тестирование классов

Ход выполнения:

- Создан класс Main вне пакета functions
- Протестированы все методы на примере функции x^2
- Проверены граничные случаи