Лабораторная работа №2

Выполнила: Аксенова Алина Владимировна
Номер группы: 6204-010302D
Дата выполнения: 2025г.

Оглавление

Задание 1

Для начала я создала пакет functions. Для этого в командной строке выполнила команду `mkdir functions`, которая создала отдельную папку для всех классов, связанных с работой с функциями. После создания папки я проверила ее наличие с помощью команды `ls`, чтобы убедиться, что директория создалась корректно.

```
mkdir functions
ls -la
```

Задание 2

Для реализации класса FunctionPoint я создала файл FunctionPoint.java в папке functions. В этом классе я объявила приватные поля x и y для хранения координат точки, что обеспечивает инкапсуляцию данных. Я реализовала три конструктора: основной конструктор принимает конкретные значения x и y, конструктор копирования создает новый объект на основе существующей точки, а конструктор по умолчанию инициализирует точку с координатами (0, 0). Также я добавила геттеры и сеттеры для доступа к полям класса, обеспечивая контролируемое изменение данных.

```
package functions;

public class FunctionPoint {
    private double x;
    private double y;

    public FunctionPoint(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public FunctionPoint(FunctionPoint point) {
        this.x = point.x;
        this.y = point.y;
    }
```

```
    public FunctionPoint() {
        this(0, 0);
    }

    public double getX() { return x; }
    public double getY() { return y; }
    public void setX(double x) { this.x = x; }
    public void setY(double y) { this.y = y; }
}
```

Задание 3

При создании класса TabulatedFunction я использовала массив объектов FunctionPoint для хранения точек функции. Я реализовала два конструктора: первый принимает левую и правую границы области определения и количество точек, равномерно распределяя их на интервале с нулевыми значениями функции; второй конструктор принимает массив значений и создает точки с соответствующими y-координатами. В обоих случаях я обеспечила упорядоченность точек по возрастанию x-координаты.

```
package functions;

public class TabulatedFunction {
    private FunctionPoint[] points;
    private int pointsCount;

    public TabulatedFunction(double leftX, double rightX, int
pointsCount) {
        if (pointsCount < 2) pointsCount = 2;
        this.pointsCount = pointsCount;
        this.points = new FunctionPoint[pointsCount + 5];

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

```
    public TabulatedFunction(double leftX, double rightX, double[]
values) {
        this.pointsCount = values.length;
        this.points = new FunctionPoint[pointsCount + 5];

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

Задание 4

Я разработала методы для работы с табулированной функцией: `getLeftDomainBorder` возвращает x-координату первой точки, `getRightDomainBorder` - последней точки. Наиболее сложным был метод `getFunctionValue`, который вычисляет значение функции в произвольной точке с помощью линейной интерполяции. Согласно комментарию преподавателя, я улучшила метод, добавив проверку точных совпадений с узлами табуляции через машинный эпсилон. Теперь метод сначала проверяет, совпадает ли x с x1 или x2 (с точностью до 1e-10), и в таких случаях возвращает соответствующий y. Если точного совпадения нет, применяется линейная интерполяция между соседними точками. Если точка находится вне области определения, метод возвращает `Double.NaN`.

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

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

public double getFunctionValue(double x) {
    final double EPSILON = 1e-10; // Машинный эпсилон для
```

```
сравнения double

    if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
        return Double.NaN;
    }

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

        // Проверяем точное совпадение с x1
        if (Math.abs(x - x1) < EPSILON) {
            return points[i].getY();
        }

        // Проверяем точное совпадение с x2
        if (Math.abs(x - x2) < EPSILON) {
            return points[i + 1].getY();
        }

        // Проверяем попадание в интервал (x1, x2)
        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;
}
```

Задание 5

Я создала комплекс методов для управления точками функции. Метод getPointsCount возвращает количество точек, getPoint возвращает копию точки по индексу (для соблюдения инкапсуляции). Метод setPoint заменяет точку, но только если новая x-координата находится между соседними точками. Также я реализовала отдельные методы getPointX, getPointY, setPointX, setPointY для работы с координатами

точек. Особое внимание уделила проверкам корректности индексов и сохранению упорядоченности точек.

```
public int getPointsCount() { return pointsCount; }

public FunctionPoint getPoint(int index) {
    if (index < 0 || index >= pointsCount) return null;
    return new FunctionPoint(points[index]);
}

public void setPoint(int index, FunctionPoint point) {
    if (index < 0 || index >= pointsCount) return;
    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) {
    if (index < 0 || index >= pointsCount) return Double.NaN;
    return points[index].getX();
}

public void setPointX(int index, double x) {
    FunctionPoint temp = new FunctionPoint(x, getPointY(index));
    setPoint(index, temp);
}

public double getPointY(int index) {
    if (index < 0 || index >= pointsCount) return Double.NaN;
    return points[index].getY();
}

public void setPointY(int index, double y) {
    if (index < 0 || index >= pointsCount) return;
    points[index].setY(y);
}
```

Задание 6

Для динамического изменения структуры функции я реализовала методы deletePoint и addPoint. Метод deletePoint удаляет точку по индексу, сдвигая остальные точки и уменьшая счетчик. Метод addPoint находит правильную позицию для вставки новой точки, чтобы сохранить порядок по x, при необходимости расширяет массив с помощью System.arraycopy. Я предусмотрела, что массив расширяется только когда действительно заполнен, что оптимизирует использование памяти.

```
public void deletePoint(int index) {
    if (index < 0 || index >= pointsCount || pointsCount <= 2)
return;

    for (int i = index; i < pointsCount - 1; i++) {
        points[i] = points[i + 1];
    }
    pointsCount--;
}

public void addPoint(FunctionPoint point) {
    int insertIndex = 0;
    while (insertIndex < pointsCount && points[insertIndex].getX()
< point.getX()) {
        insertIndex++;
    }

    if (pointsCount >= points.length) {
        FunctionPoint[] newPoints = new
FunctionPoint[points.length + 10];
        System.arraycopy(points, 0, newPoints, 0, pointsCount);
        points = newPoints;
    }

    for (int i = pointsCount; i > insertIndex; i--) {
        points[i] = points[i - 1];
    }

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

Задание 7

Для тестирования созданных классов я разработала программу Main, которая демонстрирует работу всех реализованных методов. Я создала табулированную функцию f(x) = x² на интервале [0, 4], протестировала вычисление значений в различных точках (включая точки вне области определения), добавила новые точки и удалила существующие. Программа выводит подробную информацию о функции, результаты интерполяции и изменения после модификации точек. Все тесты подтвердили корректность работы реализованных алгоритмов.

```
import functions.*;

public class Main {
    public static void main(String[] args) {
        System.out.println("=== Лабораторная работа №2: Табулированные
функции ===\n");

        // Тест 1: Создание функции f(x) = x^2
        System.out.println("1. Создание функции f(x) = x^2 на [0, 4] с 5
точками:");
        TabulatedFunction func1 = new TabulatedFunction(0, 4, 5);

        for (int i = 0; i < func1.getPointsCount(); i++) {
            double x = func1.getPointX(i);
            func1.setPointY(i, x * x);
        }

        printFunctionInfo(func1, "func1");

        // Тест 2: Проверка точных совпадений с узлами табуляции
        System.out.println("\n2. Проверка точных совпадений с узлами
табуляции:");
        System.out.println("f(0.0) = " + func1.getFunctionValue(0.0));
        System.out.println("f(1.0) = " + func1.getFunctionValue(1.0));
        System.out.println("f(2.0) = " + func1.getFunctionValue(2.0));
        System.out.println("f(3.0) = " + func1.getFunctionValue(3.0));
        System.out.println("f(4.0) = " + func1.getFunctionValue(4.0));

        // Тест 3: Проверка интерполяции
        System.out.println("\n3. Проверка интерполяции в разных точках:");
        double[] testPoints = {-1, 0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 5};

        for (double x : testPoints) {
            double y = func1.getFunctionValue(x);
            System.out.printf("f(%4.1f) = ", x);
```

```
            if (Double.isNaN(y)) {
                System.out.println("не определена");
            } else {
                System.out.printf("%6.3f\n", y);
            }
        }
    }

    public static void printFunctionInfo(TabulatedFunction func, String
name) {
        System.out.println("Функция " + name + ":");
        System.out.println("  Область определения: [" +
func.getLeftDomainBorder() + ", " + func.getRightDomainBorder() + "]");
        System.out.println("  Количество точек: " + func.getPointsCount());
        System.out.println("  Точки функции:");
        for (int i = 0; i < func.getPointsCount(); i++) {
            System.out.printf("    [%d] (%4.1f; %4.1f)\n", i,
func.getPointX(i), func.getPointY(i));
        }
    }
}
```

Результат работы Main:

```
=== Лабораторная работа №2: Табулированные функции ===

1. Создание функции f(x) = x^2 на [0, 4] с 5 точками:
Функция func1:
  Область определения: [0.0, 4.0]
  Количество точек: 5
  Точки функции:
    [0] ( 0,0;  0,0)
    [1] ( 1,0;  1,0)
    [2] ( 2,0;  4,0)
    [3] ( 3,0;  9,0)
    [4] ( 4,0; 16,0)

2. Проверка точных совпадений с узлами табуляции:
f(0.0) = 0.0
f(1.0) = 1.0
f(2.0) = 4.0
f(3.0) = 9.0
f(4.0) = 16.0

3. Проверка интерполяции в разных точках:
f(-1,0) = не определена
f( 0,0) =  0,000
f( 0,5) =  0,500
f( 1,0) =  1,000
f( 1,5) =  2,500
f( 2,5) =  6,500
f( 3,0) =  9,000
f( 3,5) = 12,500
f( 4,0) = 16,000
f( 5,0) = не определена
PS C:\Users\Alina>
```

Обновленный Main с проверкой методов добавления, удаления и замены точек:

```
in   main
=== Лабораторная работа №2: Табулированные функции ===

1. Создание функции f(x) = x^2 на [0, 4] с 5 точками:
Функция func1:
  Область определения: [0.0, 4.0]
  Количество точек: 5
  Точки функции:
    [0] ( 0,0;  0,0)
    [1] ( 1,0;  1,0)
    [2] ( 2,0;  4,0)
    [3] ( 3,0;  9,0)
    [4] ( 4,0; 16,0)

2. Проверка точных совпадений с узлами табуляции:
f(0.0) = 0.0
f(1.0) = 1.0
f(2.0) = 4.0
f(3.0) = 9.0
f(4.0) = 16.0

3. Проверка интерполяции в разных точках:
f(-1,0) = не определена
f( 0,0) =  0,000
f( 0,5) =  0,500
f( 1,0) =  1,000
f( 1,5) =  2,500
f( 2,0) =  4,000
f( 2,5) =  6,500
f( 3,0) =  9,000
f( 3,5) = 12,500
f( 4,0) = 16,000
f( 5,0) = не определена
```

```
4. Тестирование операций с точками:

Исходные точки:
Количество точек: 5
Точки функции:
    [0] (0,0; 0,00)
    [1] (1,0; 1,00)
    [2] (2,0; 4,00)
    [3] (3,0; 9,00)
    [4] (4,0; 16,00)

4.1. Добавление точки (2.5, 6.25):
Количество точек: 6
Точки функции:
    [0] (0,0; 0,00)
    [1] (1,0; 1,00)
    [2] (2,0; 4,00)
    [3] (2,5; 6,25)
    [4] (3,0; 9,00)
    [5] (4,0; 16,00)

4.2. Добавление точки (1.5, 2.25):
Количество точек: 7
Точки функции:
    [0] (0,0; 0,00)
    [1] (1,0; 1,00)
    [2] (1,5; 2,25)
    [3] (2,0; 4,00)
    [4] (2,5; 6,25)
    [5] (3,0; 9,00)
    [6] (4,0; 16,00)
```

```
4.3. Замена точки с индексом 3 на (2.8, 7.84):
Количество точек: 7
Точки функции:
    [0] (0,0; 0,00)
    [1] (1,0; 1,00)
    [2] (1,5; 2,25)
    [3] (2,0; 4,00)
    [4] (2,5; 6,25)
    [5] (3,0; 9,00)
    [6] (4,0; 16,00)

4.4. Замена x координаты точки с индексом 2 на 1.8:
Количество точек: 7
Точки функции:
    [0] (0,0; 0,00)
    [1] (1,0; 1,00)
    [2] (1,8; 2,25)
    [3] (2,0; 4,00)
    [4] (2,5; 6,25)
    [5] (3,0; 9,00)
    [6] (4,0; 16,00)

4.5. Замена y координаты точки с индексом 4 на 20.0:
Количество точек: 7
Точки функции:
    [0] (0,0; 0,00)
    [1] (1,0; 1,00)
    [2] (1,8; 2,25)
    [3] (2,0; 4,00)
    [4] (2,5; 20,00)
    [5] (3,0; 9,00)
    [6] (4,0; 16,00)
```

```
4.6. Удаление точки с индексом 1:
Количество точек: 6
Точки функции:
    [0] (0,0; 0,00)
    [1] (1,8; 2,25)
    [2] (2,0; 4,00)
    [3] (2,5; 20,00)
    [4] (3,0; 9,00)
    [5] (4,0; 16,00)

4.7. Проверка некорректных операций:
Попытка удалить точку с неверным индексом (-1):
Количество точек до: 6, после: 6 (не изменилось)
Попытка заменить точку с нарушением порядка x:
x точки с индексом 2 до: 2.0, после: 2.0 (не изменился)
PS C:\Users\Alina>
```