

Отчет по лабораторной работе №2

Разработка классов для работы с табулированными функциями

Выполнила: Андреяшкина Мария

Группа: 6204-010302D

Оглавление

Задание 1	3
Задание 2	3
Задание 3	4
Задание 4	5
Задание 5	6
Задание 6	6
Задание 7	7
Выводы	10

Задание 1

Создание пакета functions

Создан пакет functions для организации классов программы.

Пакет обеспечивает логическую группировку связанных классов.

Задание 2

Реализация класса FunctionPoint

Разработан класс для представления точки функции с инкапсуляцией данных.

Ключевые методы:

- FunctionPoint(double x, double y) - конструктор с параметрами
- FunctionPoint(FunctionPoint point) - конструктор копирования
- FunctionPoint() - конструктор по умолчанию
- getX(), getY(), setX(), setY() - методы доступа

Исходный код:

```
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, 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

Реализация класса TabulatedFunction

Создан класс для работы с табулированными функциями. Точки хранятся в упорядоченном массиве по координате X.

Конструкторы:

- TabulatedFunction(leftX, rightX, pointsCount) - равномерная сетка
- TabulatedFunction(leftX, rightX, values) - с заданными значениями

Особенности:

Массив точек имеет запас памяти (+10 элементов) для эффективного добавления новых точек.

Исходный код:

```
package functions;

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

```

public TabulatedFunction(double leftX, double rightX, int pointsCount) {
    this.pointsCount = pointsCount;
    this.points = new FunctionPoint[pointsCount+10];

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

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

public TabulatedFunction(double leftX, double rightX, double[] values) {
    this.pointsCount = values.length;
    this.points = new FunctionPoint[pointsCount+10];

    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

Методы работы с функцией

Реализованы методы для работы с областью определения и вычисления значений:

Основные методы:

- getLeftDomainBorder() - левая граница области определения
- getRightDomainBorder() - правая граница области определения
- getFunctionValue(x) - значение функции с линейной интерполяцией

Алгоритм интерполяции:

Для точки x находится интервал $[x_1, x_2]$ и вычисляется значение по формуле:

$$y = y_1 + (y_2 - y_1) \times (x - x_1) / (x_2 - x_1)$$

Для точек вне области определения возвращается Double.NaN.

Задание 5

Методы работы с точками

Реализованы методы для доступа и модификации точек с проверкой корректности.

Методы доступа:

- `getPointsCount()` - количество точек
- `getPoint(index)` - копия точки по индексу
- `getPointX(index)`, `getPointY(index)` - координаты точки

Методы модификации:

- `setPoint(index, point)` - замена точки с проверкой порядка
- `setPointX(index, x)` - изменение X с проверкой соседей
- `setPointY(index, y)` - изменение Y

Задание 6

Изменение количества точек

Реализованы методы для динамического изменения количества точек:

Методы:

- `deletePoint(index)` - удаление точки со сдвигом массива
- `addPoint(point)` - добавление точки с сохранением порядка

Оптимизация:

При добавлении точек используется `System.arraycopy()` для эффективного копирования. Массив автоматически расширяется при необходимости.

Задание 7

Тестирование работы классов

Создан класс Main для комплексного тестирования функциональности.

Процесс тестирования:

1. Создание табулированной функции $y = x^2$ на интервале $[0, 5]$
2. Вычисление значений в различных точках
3. Тестирование модификации точек

Исходный код Main:

```
public class Main {  
    public static void main(String[] args) {  
        // Создание массива значений функции (квадратичная функция  $y = x^2$ )  
        double[] values = {0.0, 1.0, 4.0, 9.0, 16.0, 25.0};  
  
        // создание объекта табулированной функции  
        functions.TabulatedFunction func = new functions.TabulatedFunction(0.0, 5.0,  
values);  
  
        System.out.println("==== Начальное состояние функции ====");  
        func.printPoints();  
  
        // Массив тестовых точек для вычисления значений функции  
        // Включает точки внутри и вне области определения  
        double[] testPoints = {-2.0, -1.0, 0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0,  
4.5, 5.0, 6.0};  
  
        // Цикл по всем тестовым точкам  
        for (double x : testPoints) {  
            // Получение значения функции в точке x  
            double y = func.getFunctionValue(x);  
  
            // Вывод результата в формате:  $f(x) = y$   
            System.out.printf("f(%4.1f) = ", x);  
  
            // Проверка, является ли результат числом (не NaN)  
            if (!Double.isNaN(y)) {  
                // Если точка в области определения - выводим значение  
                System.out.printf("%6.3f%n", y);  
            } else {  
                // Если точка вне области определения - выводим "undefined"  
                System.out.println("undefined");  
            }  
        }  
    }  
}
```

```

    }

    System.out.println("\n==== ТЕСТИРОВАНИЕ ИЗМЕНЕНИЯ ТОЧЕК ===");

    System.out.println("1. Тест setPoint:");
    System.out.println("До: f(1.5) = " + func.getFunctionValue(1.5));
    func.setPoint(2, new functions.FunctionPoint(2.0, 10.0)); // Меняем (2,4) на
(2,10)
    System.out.println("После: f(1.5) = " + func.getFunctionValue(1.5));
    func.printPoints();

    System.out.println("2. Тест setPointX:");
    System.out.println("До: f(1.2) = " + func.getFunctionValue(1.2));
    func.setPointX(1, 1.5); // Меняем x=1 на x=1.5
    System.out.println("После: f(1.2) = " + func.getFunctionValue(1.2));
    func.printPoints();

    System.out.println("3. Тест addPoint:");
    System.out.println("До: f(2.5) = " + func.getFunctionValue(2.5));
    func.addPoint(new functions.FunctionPoint(2.5, 6.25));
    System.out.println("После: f(2.5) = " + func.getFunctionValue(2.5));
    func.printPoints();

    System.out.println("4. Тест deletePoint:");
    System.out.println("До: f(0.5) = " + func.getFunctionValue(0.5));
    func.deletePoint(0); // Удаляем первую точку (0,0)
    System.out.println("После: f(0.5) = " + func.getFunctionValue(0.5));
    func.printPoints();
}

}

```

Выход программы:

```
==== Начальное состояние функции ===
```

Текущие точки функции:

```
[0] (0,000, 0,000)
[1] (1,000, 1,000)
[2] (2,000, 4,000)
[3] (3,000, 9,000)
[4] (4,000, 16,000)
[5] (5,000, 25,000)
```

```
f(-2,0) = undefined
```

```
f(-1,0) = undefined
```

```
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( 4,5) = 20,500
f( 5,0) = 25,000
f( 6,0) = undefined
```

==== ТЕСТИРОВАНИЕ ИЗМЕНЕНИЯ ТОЧЕК ===

1. Тест setPoint:

До: $f(1.5) = 2.5$

После: $f(1.5) = 5.5$

Текущие точки функции:

```
[0] (0,000, 0,000)
[1] (1,000, 1,000)
[2] (2,000, 10,000)
[3] (3,000, 9,000)
[4] (4,000, 16,000)
[5] (5,000, 25,000)
```

2. Тест setPointX:

До: $f(1.2) = 2.8$

После: $f(1.2) = 0.7999999999999999$

Текущие точки функции:

```
[0] (0,000, 0,000)
[1] (1,500, 1,000)
[2] (2,000, 10,000)
[3] (3,000, 9,000)
[4] (4,000, 16,000)
[5] (5,000, 25,000)
```

3. Тест addPoint:

До: $f(2.5) = 9.5$

После: $f(2.5) = 6.25$

Текущие точки функции:

[0] (0,000, 0,000)

[1] (1,500, 1,000)

[2] (2,000, 10,000)

[3] (2,500, 6,250)

[4] (3,000, 9,000)

[5] (4,000, 16,000)

[6] (5,000, 25,000)

4. Тест deletePoint:

До: $f(0.5) = 0.3333333333333333$

После: $f(0.5) = \text{NaN}$

Текущие точки функции:

[0] (1,500, 1,000)

[1] (2,000, 10,000)

[2] (2,500, 6,250)

[3] (3,000, 9,000)

[4] (4,000, 16,000)

[5] (5,000, 25,000)

Результаты тестирования:

- setPoint: значение $f(1.5)$ изменилось с 2.5 на 5.5 ✓
- setPointX: значение $f(1.2)$ изменилось с 2.8 на 0.8 ✓
- addPoint: значение $f(2.5)$ стало точным (6.25) ✓
- deletePoint: точка $f(0.5)$ стала undefined после удаления ✓

Выводы

В ходе лабораторной работы успешно:

- Разработаны классы FunctionPoint и TabulatedFunction с полной инкапсуляцией данных
- Реализована линейная интерполяция для вычисления значений функции

- Обеспечена корректная работа с точками (добавление, удаление, модификация)
- Протестирована работа всех методов на примере квадратичной функции
- Продемонстрирована обработка крайних случаев (точки вне области определения)

Все классы функционируют корректно, выводя ожидаемые результаты.

Особое внимание уделено соблюдению принципов ООП и эффективности работы с памятью.