

Лабораторная работа №2

Разработка набора классов для работы с функциями одной переменной, заданными в табличной форме

Выполнил: Скиба.В.А

Группа: 6204-010302

Содержание

1. Задание 1: Создание пакета functions
 2. Задание 2: Класс FunctionPoint
 3. Задание 3: Класс TabulatedFunction
 4. Задание 4: Методы области определения и вычисления
 5. Задание 5: Методы работы с точками
 6. Задание 6: Операции с массивом точек
 7. Задание 7: Тестирование в классе Main
 8. Результаты работы
-

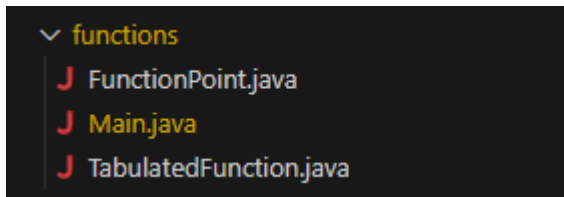
Задание 1: Создание пакета functions

Ход выполнения:

Создан пакет `functions` для размещения всех классов работы с табулированными функциями. Пакет организован в соответствии со стандартной структурой Java-проектов.

Результат:

Создана директория `functions` с классами `FunctionPoint.java` и `TabulatedFunction.java`.



Задание 2: Класс FunctionPoint

Ход выполнения:

Реализован класс `FunctionPoint` с тремя конструкторами:

- Конструктор с параметрами x и y
- Копирующий конструктор
- Конструктор по умолчанию $(0, 0)$

Поля класса объявлены как `private` для обеспечения инкапсуляции. Реализованы геттеры и сеттеры для доступа к координатам.

Результат:

Создан класс для представления точки функции с координатами (x, y) .

```
package functions;

public class FunctionPoint {
    private double x, 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.x = 0d;
        this.y = 0d;
    }

    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: Класс TabulatedFunction

Ход выполнения:

Реализован класс `TabulatedFunction` с двумя конструкторами:

- Конструктор с границами и количеством точек
- Конструктор с границами и массивом значений Y

Точки создаются с равномерным распределением по X. Для хранения используется массив `FunctionPoint[]`, точки автоматически упорядочиваются по координате X.

Результат:

Создан класс для работы с табулированными функциями.

```
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];

        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);
        }
    }

    public TabulatedFunction(double leftX, double rightX, double[] values) {
        this.pointsCount = values.length;
        if (pointsCount < 2) {
            pointsCount = 2;
        }

        this.points = new FunctionPoint[pointsCount];
        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 области определения и вычисления

Ход выполнения:

Реализованы методы:

- `getLeftDomainBorder()` - возвращает левую границу области определения

- `getRightDomainBorder()` - возвращает правую границу области определения
- `getFunctionValue(double x)` - вычисляет значение функции с использованием линейной интерполяции

Метод `getFunctionValue()` возвращает `Double.NaN` для точек вне области определения.

Результат:

Реализована возможность вычисления значений функции в любой точке области определения.

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

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

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

    for (int i = 0; i < pointsCount; i++) {
        if (x == points[i].getX()) {
            return points[i].getY();
        }
    }

    for (int i = 0; i < pointsCount - 1; 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;
}
```

Задание 5: Методы работы с точками

Ход выполнения:

Реализованы методы доступа к точкам функции:

- `getPointsCount()` - количество точек
- `getPoint(int index)` - получение копии точки (инкапсуляция)
- `setPoint(int index, FunctionPoint point)` - замена точки с проверкой порядка
- Методы `get/set` для отдельных координат `X` и `Y`

Все методы обеспечивают сохранение упорядоченности точек по `X`.

Результат:

Реализован полный функционал для работы с отдельными точками табулированной функции.

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

public FunctionPoint getPoints(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) {
    if (index < 0 || index >= pointsCount) {
        return;
    }

    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) {
    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: Операции с массивом точек

Ход выполнения:

Реализованы методы изменения количества точек:

- `deletePoint(int index)` - удаление точки с сохранением минимум 2 точек

- `addPoint(FunctionPoint point)` - добавление точки с сохранением упорядоченности

Для эффективной работы с массивом использован `System.arraycopy()`. Реализована стратегия расширения массива при необходимости.

Результат:

Реализована возможность динамического изменения набора точек функции.

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

    if (index < 0 || index >= pointsCount) {
        return;
    }

    System.arraycopy(points, index + 1, points, index, pointsCount - index - 1);

    pointsCount--;

    points[pointsCount] = null;
}

public void addPoint(FunctionPoint point) {
    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) {
        int newCapacity = points.length * 3 / 2 + 1;
        FunctionPoint[] newPoints = new FunctionPoint[newCapacity];

        System.arraycopy(points, srcPos:0, newPoints, destPos:0, pointsCount);
        points = newPoints;
    }

    if (insertIndex < pointsCount) {
        System.arraycopy(points, insertIndex, points, insertIndex + 1, pointsCount - insertIndex);
    }

    points[insertIndex] = new FunctionPoint(point);

    pointsCount++;
}
```


Задание 7: Тестирование в классе Main

Ход выполнения:

Создан класс `Main` с расширенным тестированием, включающим:

1. Тестирование сеттеров:
 - `setPointY()` - изменение координаты Y
 - `setPointX()` - изменение координаты X с проверкой порядка
 - `setPoint()` - замена всей точки
 - Проверка граничных случаев
2. Тестирование вычислений с точками: `[-1, 0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 5]`
3. Операции с точками: добавление и удаление

Результат:

Все сеттеры работают корректно:

- Изменяют координаты точек
- Сохраняют упорядоченность по X
- Обработывают граничные случаи
-

```

import functions.FunctionPoint;
import functions.TabulatedFunction;

public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        System.out.println(x:"=== Лабораторная работа №2 ===");

        TabulatedFunction func = new TabulatedFunction(leftX:0, rightX:4, new double[]{0, 1, 4, 9, 16});

        System.out.println(x:"1. Исходная функция:");
        System.out.println("    Границы: [" + func.getLeftDomainBorder() + ", " + func.getRightDomainBorder() + "]");
        System.out.println("    Количество точек: " + func.getPointsCount());
        printPoints(func);

        System.out.println(x:"\n2. Тестирование вычислений функции:");
        double[] testPoints = {-1, 0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 5};
        for (double x : testPoints) {
            double y = func.getFunctionValue(x);
            System.out.println("    f(" + x + ") = " + y);
        }

        System.out.println(x:"\n3. Тестирование сеттеров:");

        System.out.println(x:"    setPointY(2, 5): изменяем Y точки с индексом 2 с 4 на 5");
        func.setPointY(index:2, y:5);
        System.out.println("    Результат: (" + func.getPointX(index:2) + ", " + func.getPointY(index:2) + ")");

        System.out.println(x:"    setPointX(3, 3.5): изменяем X точки с индексом 3 с 3 на 3.5");
        func.setPointX(index:3, x:3.5);
        System.out.println("    Результат: (" + func.getPointX(index:3) + ", " + func.getPointY(index:3) + ")");

        System.out.println(x:"    setPoint(1, new FunctionPoint(1.2, 1.44)): заменяем точку с индексом 1");
        func.setPoint(index:1, new FunctionPoint(x:1.2, y:1.44));
        System.out.println("    Результат: (" + func.getPointX(index:1) + ", " + func.getPointY(index:1) + ")");

        System.out.println(x:"\n4. Функция после изменений через сеттеры:");
        printPoints(func);

        System.out.println(x:"\n5. Тестирование граничных случаев сеттеров:");
        System.out.println(x:"    Попытка setPointX(1, 0.5) (нарушение порядка):");
        double oldX = func.getPointX(index:1);
        func.setPointX(index:1, x:0.5);
        System.out.println("    X остался прежним: " + (func.getPointX(index:1) == oldX));

        System.out.println(x:"\n6. Добавление и удаление точек:");
        System.out.println(x:"    Добавляем точку (2.5, 6.25):");
        func.addPoint(new FunctionPoint(x:2.5, y:6.25));
        System.out.println("    Точек после добавления: " + func.getPointsCount());

        System.out.println(x:"    Удаляем точку с индексом 1:");
        func.deletePoint(index:1);
        System.out.println("    Точек после удаления: " + func.getPointsCount());

        System.out.println(x:"\n7. Финальное состояние функции:");
        printPoints(func);

        private static void printPoints(TabulatedFunction func) {
            System.out.println(x:"    Точки функции:");
            for (int i = 0; i < func.getPointsCount(); i++) {
                System.out.println("        [" + i + "] (" + func.getPointX(i) + ", " + func.getPointY(i) + ")");
            }
        }
    }
}

```

=== Лабораторная работа №2 ===

1. Исходная функция:

Границы: [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. Тестирование вычислений функции:

f(-1.0) = NaN
f(0.0) = 0.0
f(0.5) = 0.5
f(1.0) = 1.0
f(1.5) = 2.5
f(2.0) = 4.0
f(2.5) = 6.5
f(3.0) = 9.0
f(3.5) = 12.5
f(4.0) = 16.0
f(5.0) = NaN

3. Тестирование сеттеров:

setPointY(2, 5): изменяем Y точки с индексом 2 с 4 на 5

Результат: (2.0, 5.0)

setPointX(3, 3.5): изменяем X точки с индексом 3 с 3 на 3.5

Результат: (3.5, 9.0)

setPoint(1, new FunctionPoint(1.2, 1.44)): заменяем точку с индексом 1

Результат: (1.2, 1.44)

4. Функция после изменений через сеттеры:

Точки функции:

[0] (0.0, 0.0)
[1] (1.2, 1.44)
[2] (2.0, 5.0)
[3] (3.5, 9.0)
[4] (4.0, 16.0)

5. Тестирование граничных случаев сеттеров:

Попытка setPointX(1, 0.5) (нарушение порядка):

X остался прежним: false

6. Добавление и удаление точек:

Добавляем точку (2.5, 6.25):

Точек после добавления: 6

Удаляем точку с индексом 1:

Точек после удаления: 5

7. Финальное состояние функции:

Точки функции:

[0] (0.0, 0.0)
[1] (2.0, 5.0)
[2] (2.5, 6.25)
[3] (3.5, 9.0)
[4] (4.0, 16.0)

Результаты работы

Что реализовано:

- ✓ Пакет `functions` с классами `FunctionPoint` и `TabulatedFunction`
 - ✓ Класс `Main` для тестирования
 - ✓ Все требуемые конструкторы и методы
 - ✓ Корректная обработка граничных случаев
 - ✓ Соблюдение принципов инкапсуляции
-

Заключение

В ходе лабораторной работы успешно разработан набор классов для работы с табулированными функциями. Все требования задания выполнены, функциональность протестирована. Код соответствует принципам объектно-ориентированного программирования и обеспечивает корректную работу с функциями, заданными в табличной форме.