

Лабораторная работа 2

Выполнила: Рапп Ксения
Александровна
Группа: 6204-010302D

Оглавление

Задание 1: Создание пакета functions.....	3
Задание 2: Класс FunctionPoint.....	3
Задание 3: Класс TabulatedFunction.....	4
Задание 4: Методы работы с функцией.....	5
Задание 5: Методы работы с точками.....	6-7
Задание 6: Изменение количества точек.....	8
Задание 7: Тестирование классов.....	9-10

Задание 1:

Создала пакет functions для размещения классов работы с функциями. В пакете созданы два класса: FunctionPoint и TabulatedFunction.

Задание 2: Класс FunctionPoint

Реализовала класс для представления точки функции с координатами x и y.

Конструкторы:

- FunctionPoint(double x, double y) - создание точки с заданными координатами
- FunctionPoint(FunctionPoint point) - копирование существующей точки
- FunctionPoint() - создание точки (0, 0)

Геттеры и сеттеры для полей x и y обеспечивают принцип инкапсуляции.

```
package functions;
public class FunctionPoint {
    // Приватные поля для инкапсуляции данных
    private double x; // Координата x точки
    private double y; // Координата y точки
    // КОНСТРУКТОРЫ
    public FunctionPoint(double x, double y) { //Создаёт объект точки с заданными
координатами
        this.x = x;
        this.y = y;
    }
    public FunctionPoint(FunctionPoint point) {
        this.x = point.x; // Копируем x из переданной точки
        this.y = point.y; // Копируем y из переданной точки
    }
    public FunctionPoint() { //Создаёт точку с координатами (0; 0)
        this(0.0, 0.0);
    }
    // ГЕТТЕРЫ (методы для чтения значений)
    public double getX() {
        return x;
    }
    public double getY() {
        return y;
    }
    // СЕТТЕРЫ (методы для изменения значений)
    public void setX(double x) {
        this.x = x;
    }
    public void setY(double y) {
        this.y = y;
    }
}
```

Задание 3: класс TabulatedFunction

Реализовала класс для хранения табулированной функции в виде упорядоченного массива точек.

Конструкторы:

- TabulatedFunction(double leftX, double rightX, int pointsCount) - создание с указанным количеством точек
- TabulatedFunction(double leftX, double rightX, double[] values) - создание с заданными значениями y

Точки создаются через равные интервалы, массив имеет запас места для будущих добавлений точек.

```
public TabulatedFunction(double leftX, double rightX, int pointsCount) {  
    // Проверка параметров  
    if (pointsCount < 2) {  
        pointsCount = 2; // Минимум 2 точки для функции }  
    // Если границы перепутаны - меняем местами  
    if (leftX > rightX) {  
        double temp = leftX;  
        leftX = rightX;  
        rightX = temp; }  
    this.pointsCount = pointsCount;  
    // Создаем массив с запасом места для будущих добавлений  
    this.points = new FunctionPoint[pointsCount + 2];  
    // Вычисляем шаг между точками  
    double step = (rightX - leftX) / (pointsCount - 1);  
    // Создаем точки с равными интервалами  
    for (int i = 0; i < pointsCount; i++) {  
        double x = leftX + i * step; // Вычисляем x координату  
        points[i] = new FunctionPoint(x, 0.0); // y = 0 по умолчанию } }  
    // Вместо количества точек получает значения функции в виде массива  
public TabulatedFunction(double leftX, double rightX, double[] values) {  
    this.pointsCount = values.length;  
    this.points = new FunctionPoint[pointsCount + 2];  
    double step = (rightX - leftX) / (pointsCount - 1);  
    // Создаем точки с заданными значениями y  
    for (int i = 0; i < pointsCount; i++) {  
        double x = leftX + i * step;  
        points[i] = new FunctionPoint(x, values[i]); // Используем переданные значения y  
    }  
}
```

Задание 4: Методы работы с функцией

В классе `TabulatedFunction` описала методы, необходимые для работы с функцией.

Метод `double getLeftDomainBorder()` возвращает значение левой границы области определения табулированной функции:

```
public double getLeftDomainBorder() {  
    return points[0].getX(); // Первая точка - самая левая  
}
```

Метод `double getRightDomainBorder()` возвращает значение правой границы области определения табулированной функции:

```
public double getRightDomainBorder() {  
    return points[pointsCount - 1].getX(); // Последняя точка - самая правая  
}
```

Метод `double getFunctionValue(double x)` возвращает значение функции в точке `x`, если эта точка лежит в области определения функции.

В противном случае метод возвращает значение неопределённости.

При расчёте значения функции использую линейную интерполяцию.

Для написания кода метода воспользовалась уравнением прямой, проходящей через две заданные различающиеся точки.

```
public double getFunctionValue(double x) {  
    // Проверка что x в области определения  
    if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {  
        return Double.NaN; // Не число - точка вне области определения  
    }  
    // Ищем интервал, в который попадает x  
    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();  
            //уравнение прямой, проходящей через две точки  
            return y1 + (y2 - y1) * (x - x1) / (x2 - x1);  
        }  
    }  
    return Double.NaN;  
}
```

Задание 5 Методы работы с точками

В классе `TabulatedFunction` описала методы, необходимые для работы с точками табулированной функции.

Метод `int getPointsCount()` возвращает количество точек.

```
public int getPointsCount() {
    return pointsCount;
}
// Возвращает копию точки (для инкапсуляции)
public FunctionPoint getPoint(int index) {
    if (index < 0 || index >= pointsCount) {
        return null;
    }
    return new FunctionPoint(points[index]);
}
```

Метод `FunctionPoint getPoint(int index)` возвращает *копию* точки, соответствующей переданному индексу.

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

Метод `void setPoint(int index, FunctionPoint point)` заменяет указанную точку табулированной функции на переданную.

Для корректной инкапсуляции заменила на копию переданной точки.

```
public void setPoint(int index, FunctionPoint point) {
    if (index < 0 || index >= pointsCount) {
        return;
    }
    // Проверяем, что новая x координата находится между соседями
    if (index > 0 && point.getX() <= points[index - 1].getX()) {
        return;
    }
    if (index < pointsCount - 1 && point.getX() >= points[index + 1].getX()) {
        return;
    }
    // Заменяем координаты точки (создаем копию)
    points[index].setX(point.getX());
    points[index].setY(point.getY());
}
```

Метод `double getPointX(int index)` возвращает значение абсциссы точки

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

Метод `void setPointX(int index, double x)` изменяет значение абсциссы точки с указанным номером.

```
public void setPointX(int index, double x) {  
    if (index < 0 || index >= pointsCount) {  
        return;  
    }  
}
```

Метод `double getPointY(int index)` возвращает значение ординаты точки с указанным номером.

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

Метод `void setPointY(int index, double y)` должен изменять значение ординаты точки с указанным номером.

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

Задание 6: Изменение количества точек

В классе `TabulatedFunction` описала методы, изменяющие количество точек табулированной функции.

Метод `void deletePoint(int index)` удаляет заданную точку табулированной функции.

```
public void deletePoint(int index) {  
    // Нельзя удалять если точек меньше 3 или индекс неверный  
    if (index < 0 || index >= pointsCount || pointsCount <= 2) {  
        return;  
    }  
    // Сдвигаем все элементы после удаляемой точки влево  
    System.arraycopy(points, index + 1, points, index, pointsCount - index - 1);  
    pointsCount--; // Уменьшаем счетчик точек  
}
```

Метод `void addPoint(FunctionPoint point)` добавляет новую точку табулированной функции. Для копирования участков массивов воспользовалась методом `arraycopy()` класса `System`.

```
public void addPoint(FunctionPoint point) {  
    int insertIndex = 0;  
    while (insertIndex < pointsCount && points[insertIndex].getX() < point.getX()) {  
        insertIndex++;  
    }  
    // Проверяем, нет ли точки с таким же x  
    if (insertIndex < pointsCount && points[insertIndex].getX() == point.getX()) {  
        return;  
    }  
    // Проверяем нужно ли увеличивать массив  
    if (pointsCount >= points.length) {  
        // Создаем новый массив в 2 раза больше  
        FunctionPoint[] newPoints = new FunctionPoint[points.length * 2];  
        System.arraycopy(points, 0, newPoints, 0, pointsCount);  
        points = newPoints;  
    }  
    // Сдвигаем элементы чтобы освободить место для новой точки  
    System.arraycopy(points, insertIndex, points, insertIndex + 1, pointsCount - insertIndex);  
    // Вставляем новую точку (создаем копию для инкапсуляции)  
    points[insertIndex] = new FunctionPoint(point);  
    pointsCount++;  
}
```


Задание 7: Тестирование классов, а также создан класс Main для тестирования функциональности.

Создала класс Main (вне пакета functions), содержащий точку входа программы.

В методе main() создала экземпляр класса TabulatedFunction.

Результаты выполнения программы:

Тестируется табулированной ф-ции

1. Создание функции $f(x) = x^2$ на интервале $[0, 4]$ с 5 точками:

Область определения: $[0.0, 4.0]$

Количество точек: 5

Точки функции:

0: (0,00, 0,0000)

1: (1,00, 1,0000)

2: (2,00, 4,0000)

3: (3,00, 9,0000)

4: (4,00, 16,0000)

2. Вычисление значений функции в различных точках:

$f(-1,0)$ = вне области определения

$f(0,0) = 0,0000$

$f(0,5) = 0,5000$

$f(1,0) = 1,0000$

$f(1,5) = 2,5000$

$f(2,0) = 4,0000$

$f(2,5) = 6,5000$

$f(3,0) = 9,0000$

$f(3,5) = 12,5000$

$f(4,0) = 16,0000$

$f(5,0)$ = вне области определения

3. Изменение точки с индексом 2:

До изменения: (2.0, 4.0)

После изменения: (2.0, 3.0)

4. Добавление точки (1.5, 2.25):

Область определения: $[0.0, 4.0]$

Количество точек: 6

Точки функции:

0: (0,00, 0,0000)

- 1: (1,00, 1,0000)
- 2: (1,50, 2,2500)
- 3: (2,00, 3,0000)
- 4: (3,00, 9,0000)
- 5: (4,00, 16,0000)

5. Удаление точки с индексом 1:

Область определения: [0.0, 4.0]

Количество точек: 5

Точки функции:

- 0: (0,00, 0,0000)
- 1: (1,50, 2,2500)
- 2: (2,00, 3,0000)
- 3: (3,00, 9,0000)
- 4: (4,00, 16,0000)

6. Проверка значений после всех изменений:

$f(0,5) = 0,7500$

$f(1,0) = 1,5000$

$f(1,5) = 2,2500$

$f(2,0) = 3,0000$

Тестирование завершено

Вывод: Все классы успешно реализованы и протестированы.

Табулированная функция корректно работает с точками, выполняет линейную интерполяцию и поддерживает операции добавления/удаления точек с сохранением упорядоченности.