Лабораторная работа №1

Выполнила: Джумашева Камилла Исфатовна

Студент 6203-010302D группы

Ход выполнения

Задание 2

В пакете functions был разработан класс FunctionPoint, предназначенный для представления точек функции. Класс содержит два приватных поля, хранящих координаты точки по осям абсцисс и ординат.

Были реализованы три конструктора. Основной конструктор принимает значения координат x и y в качестве параметров. Конструктор копирования создает новый объект на основе существующей точки. Конструктор по умолчанию инициализирует точку с координатами (0, 0).

Для обеспечения доступа к приватным полям были добавлены методы getX() и getY(), возвращающие соответствующие значения координат. Данные методы позволяют соблюсти принцип инкапсуляции, предоставляя контролируемый доступ к данным объекта.

Задание 3

В пакете functions был создан класс TabulatedFunction для работы с табулированными функциями. Класс включает два приватных поля: массив объектов FunctionPoint для хранения точек функции и переменную для отслеживания количества точек.

Реализовано два конструктора. Первый конструктор принимает левую и правую границы области определения и количество точек. Внутри конструктора вычисляется шаг между точками, после чего создается массив точек с равномерным распределением по оси X. Значения Y при этом инициализируются нулями.

Второй конструктор также принимает границы области определения, но вместо количества точек получает массив значений функции. Аналогичным образом создаются точки с равномерным распределением по X, однако значения Y берутся из переданного массива.

Задание 4

Здесь нами были реализованы методы для работы с областью определения и вычисления значений функции. Методы getLeftDomainBorder() и getRightDomainBorder() возвращают границы области определения функции. Для этого мы обращаемся к крайним точкам массива: левая граница соответствует точке с индексом 0, правая - точке с последним индексом.

Метод getFunctionValue(double x) вычисляет значение функции в заданной точке. Вначале выполняется проверка принадлежности точки области определения функции. Если точка находится вне интервала, возвращается специальное значение Double.NaN. Реализация алгоритма вычисления значения представлена на рисунке 1.

```
// должен возвращать значение функции в точке x
public double getFunctionValue(double x) {
    if ((x > getRightDomainBorder()) || (x < getLeftDomainBorder()))
        return Double.NaN;

    if (x == arrayOfPoints[0].getX())
        return arrayOfPoints[0].getY();

    for (int i = 1; i < pointCount; i++) {
        if (arrayOfPoints[i].getX() == x)
            return arrayOfPoints[i].getY();

        if (arrayOfPoints[i].getX() > x) {
            return linearInterpolation(x,
                    arrayOfPoints[i - 1].getX(),
                    arrayOfPoints[i - 1].getY(),
                    arrayOfPoints[i].getX(),
                    arrayOfPoints[i].getY());
        }
    }

    return Double.NaN;
}
```

Рис. 1

Для точек, принадлежащих области определения, осуществляется поиск интервала, в который попадает заданная координата x. При нахождении соответствующего сегмента между двумя узловыми точками (X1, Y1) и (X2, Y2) применяется линейная интерполяция. Расчет производится по уравнению прямой, проходящей через две точки.

Наибольшую сложность при реализации представляла организация корректного поиска интервала для произвольной точки и обеспечение точности вычислений при линейной интерполяции.

Задание 5

Мною были разработаны методы для работы с отдельными точками табулированной функции. Метод getPointsCount() возвращает текущее количество точек в функции. Для получения копии конкретной точки реализован метод getPoint(int index), который создает новый объект FunctionPoint с такими же координатами, что и исходная точка.

Методы getPointX(int index) и getPointY(int index) предоставляют доступ к координатам точки по заданному индексу. Они используют соответствующие методы доступа класса FunctionPoint.

Метод setPoint(int index, FunctionPoint point) выполняет замену точки с сохранением порядка точек. Перед заменой проверяется, что новая точка не нарушает упорядоченность массив. Алгоритм проверки корректности замены точки представлен на рисунке 2.

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

    if (pointCount == 1) {
        arrayOfPoints[index].setX(x);
        return;
    }

    if (index == 0 && arrayOfPoints[1].getX() > x) {
        arrayOfPoints[index].setX(x);
    } else if (index == pointCount - 1 && arrayOfPoints[pointCount - 2].getX() < x) {
        arrayOfPoints[index].setX(x);
    } else if (index > 0 && index < pointCount - 1 &&
            arrayOfPoints[index - 1].getX() < x && arrayOfPoints[index + 1].getX() > x) {
        arrayOfPoints[index].setX(x);
    }
}
```

Рис. 2

Методы setPointX(int index, double x) и setPointY(int index, double y) изменяют отдельные координаты точки. Изменение абсциссы также включает проверку на сохранение порядка точек в массиве, в то время как изменение ординаты выполняется без дополнительных проверок.

Особую сложность представляла реализация корректных проверок при изменении координат точек, чтобы обеспечить неизменность упорядоченного состояния массива.

Задание 6

Были реализованы методы для изменения количества точек табулированной функции. Метод deletePoint(int index) осуществляет удаление точки по заданному индексу. При корректности указанного индекса выполняется сдвиг элементов массива влево с последующим уменьшением счетчика точек.

Метод addPoint(FunctionPoint point) обеспечивает добавление новой точки с сохранением порядка следования по координате X. Реализацию смотртреть на рисунке 3.

```
// добавляем точку
public void addPoint(FunctionPoint point) {
    FunctionPoint newPoint = new FunctionPoint(point);

    int insertIndex = 0;
    while (insertIndex < pointCount && arrayOfPoints[insertIndex].getX() < newPoint.getX()) {
        insertIndex++;
    }

    if (insertIndex < pointCount && arrayOfPoints[insertIndex].getX() == newPoint.getX()) {
        return;
    }

    if (pointCount == arrayOfPoints.length) {
        FunctionPoint[] newArray = new FunctionPoint[arrayOfPoints.length * 2 + 1];
        System.arraycopy(arrayOfPoints, srcPos:0, newArray, destPos:0, pointCount);
        arrayOfPoints = newArray;
    }

    if (pointCount - insertIndex > 0) {
        System.arraycopy(arrayOfPoints, insertIndex, arrayOfPoints, insertIndex + 1, pointCount - insertIndex);
    }

    arrayOfPoints[insertIndex] = newPoint;
    pointCount++;
}
```

Рис. 3

Основные трудности были связаны с обеспечением эффективного управления памятью и оптимизацией операций вставки и удаления элементов. Для этого пришлось переписать большую часть кода, т.к. С первого раза прочитать тз правильно было слишком сложной задачей для меня.

Задание 7

Для тестирования классов был создан класс Main. В методе main() выполнялась проверка функциональности табулированной функции на интервале [0, 20] с 5 точками. Значения функции устанавливались как y = 2x. Методы корректно работают и с другими значениями, но с этими было проще понять когда возникают ошибки.

Пример вывода программы представлен на рисунке 4.

```
pData\Roaming\Code\User\workspaceStorage\2b179851f682f5d0f4875e651e5e7c6c\redhat.java\jdt_
Number of points: 5
F(0.0) = 0.0
F(5.0) = 10.0
F(10.0) = 20.0
F(15.0) = 30.0
F(20.0) = 40.0
x belongs to [0.0;20.0]
x = 0.0 y = 0.0
x = 0.1 y = 0.2
x = 0.2 y = 0.4
x = 0.30000000000000004 y = 0.6000000000000001
x = 0.4 y = 0.8
x = 0.5 y = 1.0
x = 0.6 y = 1.2
x = 0.7 y = 1.4
x = 0.7999999999999999 y = 1.5999999999999999
x = 0.8999999999999999 y = 1.7999999999999998
x = 0.9999999999999999 y = 1.9999999999999998
x = 1.0999999999999999 y = 2.1999999999999997
Function [ (0.0, 0.0) (5.0, 10.0) (10.0, 20.0) (15.0, 30.0) (20.0, 40.0) ]

Set point [1] (1;2.005)
Function [ (0.0, 0.0) (1.0, 2.005) (10.0, 20.0) (15.0, 30.0) (20.0, 40.0) ]

delete point [1] (1;2.005)
Function [ (0.0, 0.0) (10.0, 20.0) (15.0, 30.0) (20.0, 40.0) ]

ddPoint point [1] (1;2.005)
Function [ (0.0, 0.0) (1.0, 2.005) (10.0, 20.0) (15.0, 30.0) (20.0, 40.0) ]

SetPointX point (1;5)
Function [ (0.0, 0.0) (5.0, 2.005) (10.0, 20.0) (15.0, 30.0) (20.0, 40.0) ]

SetPointX point (1;5)
Function [ (0.0, 0.0) (5.0, 2.005) (10.0, 20.0) (15.0, 30.0) (20.0, 40.0) ]

SetPointX point (1;5)
Function [ (0.0, 0.0) (5.0, 2.005) (10.0, 20.0) (15.0, 30.0) (20.0, 40.0) ]

SetPointY point (1;5)
Function [ (0.0, 0.0) (5.0, 5.0) (10.0, 20.0) (15.0, 30.0) (20.0, 40.0) ]
```

Рис. 4

Тяжелее всего было с setPointX() и setPointY(), которые демонстрировали различное поведение при изменении координат. Все операции выполнялись с контролем сохранения упорядоченности точек по оси X. Их реализация оказалась для меня совсем не очевидной и пришлось портратить некоторое время, чтобы из переписать пару-тройку раз.