Лабораторная работа № 2

Титульный лист

Дисциплина: Объектно-ориентированное программирование
Название работы: Разработка набора классов для работы с функциями одной переменной, заданными в табличной форме.
Выполнил: Мостовщиков Владимир Витальевич
Группа: 6204-010302D
Преподаватель: Борисов Дмитрий Сергеевич
Год: 2025

Содержание:

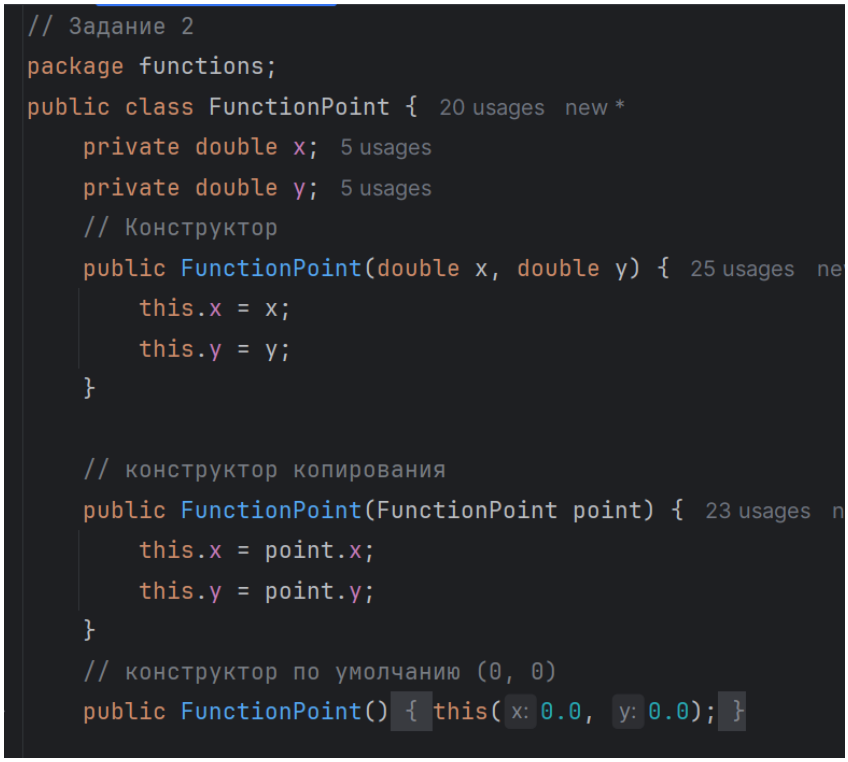
Задание 1. Создание пакета functions

Для выполнения первого задания Я запустил среду разработки intellij idea. Открыл форк репозитория, и создал пакет functions в котором выполнялись все последующие задания за исключением седьмого.

Задание 2. Создание класса FunctionPoint , конструкторов, геттеров и сеттеров

Для выполнения второго задания в пакете functions я создал класс FunctionPoint . Класс содержит приватные поля x и y для хранения координат. Инкапсуляция обеспечена с помощью геттеров и сеттеров:

- **Конструкторы:**
  - FunctionPoint(double x, double y) — создаёт точку с заданными координатами.
  - FunctionPoint(FunctionPoint point) — копирующий конструктор; создаёт точку с теми же координатами, что и у переданного объекта.
  - FunctionPoint() — конструктор по умолчанию, инициализирующий точку координатами (0, 0).
- **Геттеры/сеттеры:**
  - getX() / setX(double x) — получение/изменение абсциссы.
  - getY() / setY(double y) — получение/изменение ординаты.



```
// Задание 2
package functions;
public class FunctionPoint {  20 usages  new *
    private double x;  5 usages
    private double y;  5 usages
    // Конструктор
    public FunctionPoint(double x, double y) {  25 usages  ne
        this.x = x;
        this.y = y;
    }

    // конструктор копирования
    public FunctionPoint(FunctionPoint point) {  23 usages  n
        this.x = point.x;
        this.y = point.y;
    }
    // конструктор по умолчанию (0, 0)
    public FunctionPoint() { this( x: 0.0,  y: 0.0); }
```

Рисунок 1 – реализация конструкторов

```
        // Геттер для координаты x
    public double getX() { return x; }
        // Геттер для координаты y
    public double getY() { return y; }
        // Сеттер для координаты x
    public void setX(double x) { this.x = x; }
        // Сеттер для координаты y
    public void setY(double y) { this.y = y; }
}
```

Рисунок 2 – Реализация геттеров и сеттеров

Задание 3. Создание и реализация класса TabulatedFunction

В ходе выполнения задания я создал класс TabulatedFunction, хранящий набор точек функции и реализовал конструкторы, которые требовались в задании.

```
// Задание 3
package functions;

public class TabulatedFunction {  3 usages  new *
    private int size; // количество точек  28 usages
    private FunctionPoint[] points; // упорядоченный массив точек  39 usages

    // Табулирование на интервале [leftX; rightX] с заданным числом точек
    // Первый конструктор
    public TabulatedFunction(double leftX, double rightX, int pointsCount) {
        this.size = pointsCount;
        this.points = new FunctionPoint[pointsCount];

        // Постоянный шаг по x
        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);
        }
    }
```

```java
// Табулирование на [leftX; rightX] по массиву значений values
//Второй конструктор
public TabulatedFunction(double leftX, double rightX, double[] values) {
    this.size = values.length;  //Количество точек
    this.points = new FunctionPoint[size];

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

Рисунок 3,4 – реализация конструкторов

Задание 4. Реализация методов для работы с табулированной функцией

При выполнении задания в классе TabulatedFunction я создал методы для работы с функцией: double getLeftDomainBorder(),double getRightDomainBorder(),double getFunctionValue(double x)

```java
// Задание 4

// Левая граница области определения x
public double getLeftDomainBorder() {  1 usage   new *
    if (size == 0) {
        return Double.NaN; // Если точек нет
    }
    return points[0].getX();
}

// Правая граница области определения x
public double getRightDomainBorder() {  1 usage   new *
    if (size == 0) {
        return Double.NaN;
    }
    return points[size - 1].getX();
}
```

Рисунок 5 – реализация методов границы области определения

```java
// Метод, возвращающий значение функции в точке x с использованием линейной интерполяции
public double getFunctionValue(double x) {  2 usages  new *
    double result = Double.NaN; // значение по умолчанию вне области определения

    double left = points[0].getX();
    double right = points[size - 1].getX();

    // Должно выполняться условие что  x внутри отрезка [left, right]
    if (x >= left && x <= right) {

        // Если x совпадает с абсциссой из таблицы
        for (int i = 0; i < size; ++i) {
            if (x == points[i].getX()) {
                result = points[i].getY();
                break; // если нашли, то выходим из цикла
            }
        }

        // Если точного совпадения нет, тогда интерполируем
        if (Double.isNaN(result)) {
            for (int i = 0; i < size - 1; ++i) {
                double x0 = points[i].getX();
                double x1 = points[i + 1].getX();
                if (x >= x0 && x <= x1) {
                    double y0 = points[i].getY();
                    double y1 = points[i + 1].getY();
                    result = y0 + (y1 - y0) * (x - x0) / (x1 - x0); // линейная интерполяция
                    break;
                }
            }
        }
    }
    return result;
}
```

Рисунок 6,7 – Реализация метода getFunctionValue()

Задание 5. Реализация методов для работы с точками табулированной функции

Для выполнения данного задания в классе TabulatedFunction я описал

```
// Задание 5

    // Метод возвращает количество точек
    public int getPointsCount() { return size; }

    // Метод возвращает копию точки, соответствующей переданному индексу
    public FunctionPoint getPoint(int index) { return new FunctionPoint(points[index]); }

    // Сеттер меняющий точки с табулированной функции на переданную
    public void setPoint(int index, FunctionPoint point) {  1 usage   new *
        if (point == null)
            return;

        double newX = point.getX();

        if (index > 0) {
            double leftX = points[index - 1].getX();
            if (!(newX > leftX))
                return;
        }
```

```
        if (index < size - 1) {
            double rightX = points[index + 1].getX();
            if (!(newX < rightX))
                return;
        }

        //  Сохраняем копию переданной точки
        points[index] = new FunctionPoint(point);
    }

    // Метод, возвращающий абсциссу точки по индексу
    public double getPointX(int index) {  4 usages   new *

        if (index >= 0 && index < size) {
            return points[index].getX();
        } else {
            return Double.NaN; // если некорректный индекс
        }
    }
```

```java
// Метод, изменяющий абсциссу точки по индексу
public void setPointX(int index, double x) {  1 usage  new *
    boolean canSet = false; // заводим флаг

    // проверка, что индекс в диапазоне
    if (index >= 0 && index < size) {
        if (size == 1) {

            canSet = true;
        } else if (index == 0) {
            // если левый край, то новый x строго меньше следующей точки
            double nextX = points[1].getX();
            canSet = (x < nextX);
        } else if (index == size - 1) {
            // если правый край, то новый x строго больше предыдущей точки
            double prevX = points[size - 2].getX();
            canSet = (x > prevX);
        } else {
            //если середина, то строго между соседними точками
            double leftX = points[index - 1].getX();
            double rightX = points[index + 1].getX();
            canSet = (x > leftX && x < rightX);
```

```java
            //если середина, то строго между соседними точками
            double leftX = points[index - 1].getX();
            double rightX = points[index + 1].getX();
            canSet = (x > leftX && x < rightX);
        }

        if (canSet) {
            points[index].setX(x);
        }
    }

}

public double getPointY(int index) {return points[index].getY();}  4 usages  new *
public void setPointY(int index, double y) {points[index].setY(y);}  1 usage  new *
```

Рисунки 8,9,10,11 – Реализация всех вышеперечисленных методов

Задание 6. Реализация методов, изменяющих количество точек табулированной функции.

В данном задании я реализовал два метода класса TabulatedFunction, меняющих число точек таблицы:

```
    // Задание 6


        // Удаляет точку по индексу со сдвигом влево
       public void deletePoint(int index) {  1 usage   new *
        if (index < 0 || index >= size) return;   // если индекс вне диапазона — выходим
        System.arraycopy(points, srcPos: index + 1, points, index, length: size - index - 1);
        points[--size] = null;
    }
```

Рисунок 12 – Реализация метода deletePoint()

```
 // Добавление новой точки с сохранением порядка по x
public void addPoint(FunctionPoint point) {  1 usage   new *
 if (point == null) return;

 double x = point.getX();

 // Находим первый индекс, где points[i].x >= x и  запрещаем дубликат x
 int insertIndex = 0;
 while (insertIndex < size && points[insertIndex].getX() < x) {
     insertIndex++;
 }
 if (insertIndex < size && points[insertIndex].getX() == x) {
     return;}

 //  Расширение буфера
 if (size >= points.length) {
     int newCapacity = (points.length > 0) ? (points.length * 3 / 2 + 1) : 2; // min = 2
     FunctionPoint[] newArray = new FunctionPoint[newCapacity];
     System.arraycopy(points, srcPos: 0, newArray, destPos: 0, size);
     points = newArray;
 }
```

```
    //  Расширение буфера
    if (size >= points.length) {
        int newCapacity = (points.length > 0) ? (points.length * 3 / 2 + 1) : 2; // min = 2
        FunctionPoint[] newArray = new FunctionPoint[newCapacity];
        System.arraycopy(points, srcPos: 0, newArray, destPos: 0, size);
        points = newArray;
    }

    if (insertIndex < size) {
        System.arraycopy(points, insertIndex, points, destPos: insertIndex + 1, length: size - insertIndex);
    }
    points[insertIndex] = new FunctionPoint(point);
    size++;
}
}
```

Рисунок 13,14 – Реализация метода addPoint()

Задание 7. Проверка работы классов в Main.java

Для выполнения задания 7 создаем в пакете по умолчанию класс Main.

В классе Main (файл Main.java) в методе main() демонстрируется работа табулированной функции и всех реализованных методов.

Код можно условно разбить на несколько этапов.

1. Создание таблицы для функции y = 5x + 10 на отрезке [0; 10]

   double[] values = {10, 15, 20, …, 60};

   TabulatedFunction f = new TabulatedFunction(0.0, 10.0, values);

Затем программа выводит количество точек и границы области определения, печатает все точки таблицы.

```
C:\Users\robot\.jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program Files\Je
Шаг 1. Создаём таблицу для y = 5*x + 10 на отрезке [0; 10] с шагом 1 (11 точек).

— Границы области определения и количество точек —
Количество точек: 11
Левая граница:    0.0
Правая граница:   10.0

— Исходные точки таблицы —
точка 0: (0.0; 10.0)
точка 1: (1.0; 15.0)
точка 2: (2.0; 20.0)
точка 3: (3.0; 25.0)
точка 4: (4.0; 30.0)
точка 5: (5.0; 35.0)
точка 6: (6.0; 40.0)
точка 7: (7.0; 45.0)
точка 8: (8.0; 50.0)
точка 9: (9.0; 55.0)
точка 10: (10.0; 60.0)
```

Рисунок 15 – Результат выполнения шага 1

2. Вычисление значений функции в различных точках

В цикле выводятся значения f(x) для ряда x, включая значения вне области определения и несколько точек внутри одного интервала. При x вне области метод возвращает NaN, что и отображается.

```
Шаг 2. Проверяем вычисления f(x) в ряде точек (включая вне области):
f(-1.0) = не определена (вне области)
f(0.0) = 10.0
f(0.25) = 11.25
f(1.0) = 15.0
f(3.2) = 26.0
f(3.6) = 28.0
f(7.5) = 47.5
f(10.0) = 60.0
f(10.5) = не определена (вне области)
```

Рисунок 16 –Результат выполнения шага 2

3 Тестирование геттеров/сеттеров

Программа выводит x[2] и y[2] через геттеры, затем изменяет абсциссу и ординату у одной из точек с помощью setPointX() и setPointY(), а также заменяет целиком точку setPoint().

После каждой операции выводятся изменённые координаты и печатается обновлённый список точек

```
Меняем абсциссу точки с индексом 3 на 3.3 (строго между соседями 2 и 4).
После setPointX: x[3] = 3.3, y[3] = 25.0
Меняем ординату точки с индексом 4 на 55.0 (локально ломаем прямую).
После setPointY: x[4] = 4.0, y[4] = 55.0
Заменяем точку с индексом 5 на (5.5; 37.5) через setPoint (между 5 и 6).
После setPoint:  x[5] = 5.5, y[5] = 37.5


— Точки таблицы после правок setPointX/setPointY/setPoint —
точка 0: (0.0; 10.0)
точка 1: (1.0; 15.0)
точка 2: (2.0; 20.0)
точка 3: (3.3; 25.0)
точка 4: (4.0; 55.0)
точка 5: (5.5; 37.5)
точка 6: (6.0; 40.0)
точка 7: (7.0; 45.0)
точка 8: (8.0; 50.0)
точка 9: (9.0; 55.0)
точка 10: (10.0; 60.0)
```

Рисунок 17 – Результат выполнения шага 3

4 Добавление и удаление точек

Новая точка (7.5; 47.5) добавляется методом addPoint(), затем первая точка удаляется методом deletePoint().

После каждой операции выводится количество точек и полный список точек.

```
Шаг 4. Добавляем новую точку (7.5; 47.5), а затем удаляем самую левую точку (индекс 0).
Теперь точек: 12

— Итоговое состояние точек —
точка 0: (1.0; 15.0)
точка 1: (2.0; 20.0)
точка 2: (3.3; 25.0)
точка 3: (4.0; 55.0)
точка 4: (5.5; 37.5)
точка 5: (6.0; 40.0)
точка 6: (7.0; 45.0)
точка 7: (7.5; 47.5)
точка 8: (8.0; 50.0)
точка 9: (9.0; 55.0)
точка 10: (10.0; 60.0)
```

Рисунок 18 – Результат выполнения шага 4

5 Контрольные вычисления после модификаций

```
Шаг 5. Контрольные вычисления после всех изменений:
f(-1.0) = не определена (вне области)
f(0.0) = не определена (вне области)
f(1.0) = 15.0
f(2.5) = 21.923076923076923
f(3.5) = 33.57142857142858
f(4.0) = 55.0
f(4.5) = 49.166666666666664
f(7.4) = 47.0
f(7.5) = 47.5
f(7.6) = 48.0
f(10.0) = 60.0
f(11.0) = не определена (вне области)
```

Рисунок 19 – вывод значений после модификаций