Лабораторная работа № 2

Выполнила: Оленина Арина Игоревна
группа 6204-010302D

Оглавление

Задание 1

В среде разработки я создала папку src (папка в Java проектах, где хранится весь написанный код) и в ней пакет functions, в котором далее будут создаваться классы программы.

Задание 2

В пакете functions создала класс FunctionPoint, объект которого описывает одну точку табулированной функции.
Состояние объектов содержит аспекта: координату точки по оси абсцисс и координату точки по оси ординат.

В классе описаны следующие конструкторы:

• FunctionPoint(double x, double y) (конструктор с заданными координатами) – создаёт объект точки с заданными координатами;
• FunctionPoint(FunctionPoint point) (конструктор копирования) – создаёт объект точки с теми же координатами, что у указанной точки;
• FunctionPoint() (конструктор по умолчанию) – создаёт точку с координатами (0; 0)
Так же я добавила геттеры и сеттеры.
Результат:

```
package functions;
public class FunctionPoint {
   private double x;
   private double y;
   public FunctionPoint(double x, double y) { //Конструктор с заданными
координатами
      this.x = x;
      this.y = y;
   }
   public FunctionPoint(FunctionPoint point) { //Конструктор копирования
      this.x = point.x;
      this.y = point.y;
   }
   public FunctionPoint() { //Конструктор по умолчанию
      this(0.0, 0.0);
   }
   public double getX() {
      return x;
   }

   public void setX(double x) {
      this.x = x;
   }

   public double getY() {
      return y;
```

```
    }

    public void setY(double y) {
        this.y = y;
    }
}
```

Задание 3

В пакете functions создала класс TabulatedFunction, объект которого должен описывать табулированную функцию.
Для хранения данных о точках используется массив типа FunctionPoint. При этом работу с массивом я организовала так, чтобы точки в нём были всегда упорядочены по значению координаты x.
В классе описаны следующие конструкторы:

• TabulatedFunction(double leftX, double rightX, int pointsCount) – создаёт объект табулированной функции по заданным левой и правой границе области определения, а также количеству точек для табулирования (значения функции в точках при этом следует считать равными 0);
• TabulatedFunction(double leftX, double rightX, double[] values) – аналогичен предыдущему конструктору, но вместо количества точек получает значения функции в виде массива.
В обоих случаях точки создаются через равные интервалы по x.
Результат:

```
package functions;

public class TabulatedFunction {
    private FunctionPoint[] points; //Массив для хранения точек
    private int pointsCount; //Счетчик точек
    public TabulatedFunction(double leftX, double rightX, int pointsCount)
{ //Конструктор по количеству точек
        this.pointsCount = pointsCount;
        this.points = new FunctionPoint[pointsCount + 10]; //Запас места
        double step = (rightX - leftX) / (pointsCount - 1); //Задаем шаг
        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)
{ //Конструктор по значениям функции в виде массива
        this.pointsCount = values.length;
        this.points = new FunctionPoint[pointsCount + 10]; //Запас места
        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

В классе TabulatedFunction я описала методы, необходимые для работы с функцией.
• Метод double getLeftDomainBorder() должен возвращает значение левой границы области определения табулированной функции.
• Метод double getRightDomainBorder() возвращает значение правой границы области определения табулированной функции.
• 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 (Double.NaN) При расчёте значения функции я использовала линейную интерполяцию.
Результат:

```
public double getLeftDomainBorder() { //Метод для определения левой границы
        return points[0].getX();
    }

    public double getRightDomainBorder() { //Метод для определения правой границы
        return points[pointsCount - 1].getX();
    }
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        for (int i = 0; i < pointsCount - 1; i++) { //Ищем интервал, в котором содержится x
            double x1 = points[i].getX();
            double x2 = points[i + 1].getX();

            if (x >= x1 && x <= x2) { // Линейная интерполяция
                double y1 = points[i].getY();
                double y2 = points[i + 1].getY();
                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
            }
        }
        return Double.NaN;
    }
```

Задание 5

В классе TabulatedFunction описала методы, необходимые для работы с точками табулированной функции (нумерация точек с нуля).
• Метод int getPointsCount() возвращает количество точек.

• Метод FunctionPoint getPoint(int index) возвращает копию точки, соответствующей переданному индексу (возвращение ссылки на саму точку противоречит принципу инкапсуляции).
• Метод void setPoint(int index, FunctionPoint point) заменяет указанную точку табулированной функции на копию переданной точки. В случае если координата x задаваемой точки лежит вне интервала, замена не проводится.
• Метод double getPointX(int index) возвращает значение абсциссы точки с указанным номером.
• Метод void setPointX(int index, double x) изменяет значение абсциссы точки с указанным номером.
• Метод double getPointY(int index) возвращает значение ординаты точки с указанным номером.
• Метод void setPointY(int index, double y) изменяет значение ординаты точки с указанным номером.
Результат:

```
public int getPointsCount() { //Получение количества точек
       return pointsCount;
   }
   public FunctionPoint getPoint(int index) { // Возврат копии точки
       if (index >= 0 && index < pointsCount) {
          return new FunctionPoint(points[index]);
       }
       else{
          return null;
       }
   }
   public void setPoint(int index, FunctionPoint point) { //Замена значения
абсциссы с указанным номером
       if (index < 0 || index >= pointsCount) {
          return;
       }
       double newX = point.getX();
       if (pointsCount == 1) { // Если всего 1 точка
          points[0] = new FunctionPoint(point);
          return;
       }
       if (index == 0) { // Проверка для первого элемента
          if (newX < points[1].getX()) { // строго меньше следующего
             points[0] = new FunctionPoint(point);
          }
       }
       else if (index == pointsCount - 1) { // Проверка для последнего
элемента
          if (newX > points[pointsCount - 2].getX()) { // строго больше
предыдущего
             points[pointsCount - 1] = new FunctionPoint(point);
          }
       }
       else { // Проверка для средних элементов
```

```
            if (newX > points[index - 1].getX() && newX < points[index +
1].getX()) {
                points[index] = new FunctionPoint(point);
            }
        }
    }
    public double getPointX(int index) { //Возврат значения абсциссы с
указанным номером
        return points[index].getX();
    }

    public void setPointX(int index, double x) { //Замена значения абсциссы
с указанным номером
        points[index].setX(x);
    }

    public double getPointY(int index) { //Возврат значения ординаты с
указанным номером
        return points[index].getY();
    }

    public void setPointY(int index, double y) { //Замена значения ординаты
с указанным номером
        points[index].setY(y);
    }
```

Задание 6

В классе TabulatedFunction я описала методы, изменяющие количество точек табулированной функции.
• Метод void deletePoint(int index) удаляет заданную точку табулированной функции.
• Метод void addPoint(FunctionPoint point) добавляет новую точку табулированной функции.
Для копирования участков массивов я воспользовалась
методом arraycopy() класса System.
Результат:

```
public void deletePoint(int index) {
        if (index >= 0 && index < pointsCount) { // Сдвиг точек влево,
начиная с удаляемой
            System.arraycopy(points, index + 1, points, index, pointsCount -
index - 1);
            pointsCount--;
        }
    }
    public void addPoint(FunctionPoint point) {
        if (pointsCount >= points.length) { // Если массив заполнен,
увеличиваю его
            FunctionPoint[] newPoints = new FunctionPoint[points.length * 2];
            System.arraycopy(points, 0, newPoints, 0, pointsCount);
```

```
            points = newPoints;
        }
        int insertIndex = 0; // Нахожу позицию для вставки
        while (insertIndex < pointsCount && points[insertIndex].getX() <
point.getX()) {
            insertIndex++;
        }
        if (insertIndex < pointsCount && // Проверяю, не существует ли уже
точка с таким X
                points[insertIndex].getX() == point.getX()) {
            return; // Точка с таким X уже существует
        }
        // Сдвиг точек вправо
        System.arraycopy(points, insertIndex, points, insertIndex + 1,
pointsCount - insertIndex);
        points[insertIndex] = new FunctionPoint(point);
        pointsCount++;
    }
```

Задание 7

Затем я проверила работу написанных классов.

В пакете по умолчанию (вне пакета functions) создала класс Main, содержащий точку входа программы.
В методе main() создала экземпляр класса TabulatedFunction и задала для него табулированные значения кубической функции y = x^3. Всего 5 точек, x изменяется от -2 до 2. Вывела в консоль значения функции на ряде точек. Я использовала следующие точки: -3.0, -2.0, -1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0, 3.0 Некоторые из них не входят в область определения функции.
После компиляции и выполнения кода получила следующее:

```
Исходные точки:
x = -2.0, y = -8.0
x = -1.0, y = -1.0
x = 0.0, y = 0.0
x = 1.0, y = 1.0
x = 2.0, y = 8.0
не определено -3.0
-2.0=-8.0 ожидалось = -8.0
-1.5=-4.5 ожидалось = -3.375
-1.0=-1.0 ожидалось = -1.0
-0.5=-0.5 ожидалось = -0.125
0.0=0.0 ожидалось = 0.0
0.5=0.5 ожидалось = 0.125
1.0=1.0 ожидалось = 1.0
1.5=4.5 ожидалось = 3.375
2.0=8.0 ожидалось = 8.0
не определено 3.0
```

Затем я заменила вторую точку на точку (-1.5, -3.375), добавила точку (4.0, 64) и удалила 4 по счету точку (с координатами (1.0, 1.0).

```
После замены точки:
x = -2.0, y = -8.0
x = -1.5, y = -3.375
x = 0.0, y = 0.0
x = 1.0, y = 1.0
x = 2.0, y = 8.0
После добавления точки:
x = -2.0, y = -8.0
x = -1.5, y = -3.375
x = 0.0, y = 0.0
x = 1.0, y = 1.0
x = 2.0, y = 8.0
x = 4.0, y = 64.0
После удаления точки:
x = -2.0, y = -8.0
x = -1.5, y = -3.375
x = 0.0, y = 0.0
x = 2.0, y = 8.0
x = 4.0, y = 64.0
```

После всех преобразований снова проверила, как программа будет работать на тестовых точках:

```
не определено -3.0
-2.0=-8.0 ожидалось = -8.0
-1.5=-3.375 ожидалось = -3.375
-1.0=-2.25 ожидалось = -1.0
-0.5=-1.125 ожидалось = -0.125
0.0=0.0 ожидалось = 0.0
0.5=2.0 ожидалось = 0.125
1.0=4.0 ожидалось = 1.0
1.5=6.0 ожидалось = 3.375
2.0=8.0 ожидалось = 8.0
3.0=36.0 ожидалось = 27.0
```