

Лабораторная работа №2

Манякин Степан 6204-010302D

## Задание 1

Первым делом создаем пакет functions, в котором далее будут создаваться классы программы.

```
package functions;
```

Рис. 1

## Задание 2

В этом же пакете создаем класс FunctionPoint, объект которого должен описывать одну точку табулированной функции.




Рис. 2

С начала создаем приватные поля (для инкапсуляции наших данных)

```
public class FunctionPoint { 17 usages
    private double x; //координата по оси x 6 usages
    private double y; //координата по оси y 6 usages
```

Рис. 3

После этого создаем конструктор с двумя полями (строго по заданию)

```
public FunctionPoint(double x, double y) { //конструктор с двумя параметрами
    this.x = x;
    this.y = y;
}
```

Рис. 4

Далее создаем конструктор копирования, он создает новую точку с такими же координатами.

```
public FunctionPoint(FunctionPoint point) { //конструктор копирования
    this.x = point.x;
    this.y = point.y;
}
```

Рис. 5

Следом создаем конструктор по умолчанию, который создает точку в начале координат.

```
public FunctionPoint() { //конструктор по умолчанию (0;0)
    this.x = 0;
    this.y = 0;
}
```

Рис. 6

Далее создаем геттеры и сеттеры для координат X и Y

```
public double getX() { //геттер для x 14 usages
    return x;
}

public void setX(double x) { //сеттер для x 1 usage
    this.x = x;
}

public double getY() { //геттер для y 5 usages
    return y;
}

public void setY(double y) { //сеттер для y 1 usage
    this.y = y;
}
```

Рис. 7

На этом задание 2 завершено.

## Задание 3

Создаем новый класс TabulatedFunction.

Рис.8

По заданию нужно для хранения данных о точках должен использоваться массив типа FunctionPoint

```
public class TabulatedFunction { 3 usages
    private FunctionPoint[] points; //массив точек 32 usages
    private int pointsCount; //реальное количество точек 19 usages
```

Рис. 9

Далее создаем первый конструктор, который создаёт объект табулированной функции по заданным левой и правой границе области определения, а также количеству точек для табулирования.

```
//конструктор 1: leftX, rightX, pointsCount, y = 0
public TabulatedFunction(double leftX, double rightX, int pointsCount) {
    this.pointsCount = pointsCount;
    points = new FunctionPoint[pointsCount];

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

Рис. 10

Второй конструктор аналогичен первому, однако вместо кол-ва точек получает значения функции в виде массива.

```

public TabulatedFunction(double leftX, double rightX, double[] values) {
    this.pointsCount = values.length;
    points = new FunctionPoint[pointsCount];

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

```

Рис. 11

Задание 3 завершено.

## Задание 4

Нам надо описать методы, необходимые для работы с функцией.

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

```

public double getLeftDomainBorder() { //левая граница области определения
    return points[0].getX();
}

```

Рис. 12

Аналогично и правый:

```

public double getRightDomainBorder() { //правая граница области определения
    return points[pointsCount - 1].getX();
}

```

Рис. 13

Далее мы описываем метод `double getFunctionValue(double x)` и проверяем, входят ли координаты в границы области определения

```

public double getFunctionValue(double x) { //значение функции в точке x с линейной интерполяцией
    if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
        return Double.NaN;
    }
}

```

Рис. 14

Ищем точные совпадения.

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

Рис. 15

Далее мы проводим так называемую линейную интерполяцию. Для этого сначала ищем отрезок, который попадает x

```
for (int i = 0; i < pointsCount - 1; i++) { //линейная интерполяция между соседними точками
    double x1 = points[i].getX();
    double x2 = points[i + 1].getX();
```

Рис. 16

Проверяем, находится ли x между этих точек и пишем формулу самой интерполяции

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

Рис. 17

Если что-то пошло не так, возвращаем нечисло

```
    return Double.NaN;
}
```

Рис. 18

Задание 4 окончено.

## Задание 5

В том же классе нужно описать методы, необходимые для работы с точками табулированной функции. Данный метод просто возвращает кол-во точек.

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

Рис. 19

Далее создаем метод, который возвращает копию точки, это сделано для ее защиты от внешних изменений.

```
public FunctionPoint getPoint(int index) { //  
    return new FunctionPoint(points[index]);  
}
```

Рис. 20

Следующий метод заменяет указанную точку табулированной функции на переданную. Для корректной инкапсуляции заменяем на копию переданной точки. Если координата  $x$  задаваемой точки лежит вне интервала, определяемого значениями соседних точек табулированной функции, то замену не делаем.

```
public void setPoint(int index, FunctionPoint point) { //заменяем точку на нов  
    double newX = point.getX();  
  
    if (index > 0 && newX <= points[index - 1].getX()) return;  
    if (index < pointsCount - 1 && newX >= points[index + 1].getX()) return;  
  
    points[index] = new FunctionPoint(point);  
}
```

Рис. 21

Далее создаем метод, который возвращает значение абсциссы точки с указанным номером.

```
public double getPointX(int index) {  
    return points[index].getX();  
}
```

Рис. 22

Следующий метод изменяет значение абсциссы точки с указанным номером, если это не нарушает порядок.

```
public void setPointX(int index, double x) { //изменить x точки если не наруш  
    if (index > 0 && x <= points[index - 1].getX()) return;  
    if (index < pointsCount - 1 && x >= points[index + 1].getX()) return;  
  
    points[index].setX(x);  
}
```

Рис. 23

Далее метод возвращает значение по оси Y

```
public double getPointY(int index) { //  
    return points[index].getY();  
}
```

Рис. 24

Далее меняем значение ординаты с указанным номером

```
public void setPointY(int index, double y) { //  
    points[index].setY(y);  
}
```

Рис. 25

Задание 5 закончено

## Задание 6

Нужно описать методы, изменяющие количество точек табулированной функции.



Реализуем удаление точки. Помним, что нельзя удалять последнюю точку.

Мы используем метод `arraycopy`, так как это нативный метод джавы, который копирует данные из одного массива в другой на уровне системы. Это исключает возможные ошибки. Он рекомендуется для работы с массивами в джава

```
public void deletePoint(int index) { 1 usage
    if (pointsCount <= 1) return; // нельзя удалить последнюю точку
    |
    System.arraycopy(points, srcPos: index + 1, points, index, length: pointsCount - index - 1)

    pointsCount--; //теперь на одну точку меньше
}
```

Рис. 26

Далее необходимо реализовать добавление точки. Сначала смотрим, заполнен ли массив, если да, то увеличиваем его, создаем новый на 1 элемент больше, копируем все туда и заменяем старый на новый. Далее мы ищем место для вставки, где  $x \geq$  чем новый  $x$ . После этого освобождаем место, сдвигая все элементы вправо. Вставляем туда копию и увеличиваем счетчик. Здесь также применяется метод `arraycopy`.

```
public void addPoint(FunctionPoint point) { //добавляем новую точку с сохранением порядка x 1 usage
    if (pointsCount == points.length) { //если массив полон создаём массив большего размера
        FunctionPoint[] newPoints = new FunctionPoint[points.length + 1];
        System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, pointsCount);
        points = newPoints;
    }

    int insertIndex = 0; //находим позицию вставки сохраняем порядок по x
    while (insertIndex < pointsCount && points[insertIndex].getX() < point.getX()) {
        insertIndex++;
    }

    System.arraycopy(points, insertIndex, points, destPos: insertIndex + 1, length: pointsCount - insertIndex);
    points[insertIndex] = new FunctionPoint(point); //вставляем копию новой точки

    pointsCount++; //увеличиваем количество точек
}
```

Рис. 27

Задание 6 завершено.

## Задание 7

Проверяем работоспособность наших классов.

Создаем метод `main()`, а в нем создаем экземпляр класса `TabulatedFunction`, и задаем табулированные значения функции  $y=x^2$  на интервале  $[0, 4]$  с пятью точками.

```
import functions.TabulatedFunction;
import functions.FunctionPoint;

public class Main {
    public static void main(String[] args) {
        double[] values = {0, 1, 4, 9, 16};
```

Рис. 28

Далее создаем объект табулированной функции.

```
TabulatedFunction f = new TabulatedFunction( leftX: 0, rightX: 4, values);
```

Рис. 29

Выводим заголовок для первого раздела и проходим по значениям  $x$  от  $-1$  до  $5$  с шагом  $0.5$ . Получаем значение функции в точке по  $x$  и выводим результат. Если не число – выводим NaN

```
System.out.println("Изначальная функция y = x^2");
for (double x = -1; x <= 5; x += 0.5) {
    double y = f.getFunctionValue(x);
    System.out.printf("f(%.1f) = %s\n", x, Double.isNaN(y) ? "NaN" : y);
}
```

Рис.30

Далее мы демонстрируем работу с отдельными точками функции

Этот метод нужен для копирования точек

```
System.out.println("\nработа с точками");//работа с точками
FunctionPoint point = f.getPoint( index: 2); // копируем третью точку
System.out.println("копия точки 2: (" + point.getX() + ", " + point.getY() + ")");
```

Рис. 31

Следующий метод нужен для изменения ординаты точки

```
f.setPointY( index: 2, y: 10); //изменяем y третьей точки
System.out.println("после изменения Y точки 2: f(1.5) ≈ " + f.getFunctionValue(x: 1.5));
```

Рис. 32

Данный метод изменяет абсциссу точки.

```
f.setPointX( index: 2, x: 2.5); //перемещаем точку по X
System.out.println("после изменения X точки 2: f(2.5) = " + f.getFunctionValue(x: 2.5));
```

Рис. 33

Этот метод добавляет новую точку.

```
f.addPoint(new FunctionPoint( x: 3.2, y: 11)); //добавляем новую точку
System.out.println("после добавления точки (3.2,11): f(3.2) = " + f.getFunctionValue(x: 3.2));
```

Рис. 34

Данный метод удаляет точку.

```
f.deletePoint( index: 0); //удаляем точку
System.out.println("после удаления первой точки: f(0) = " + f.getFunctionValue(x: 0));
```

Рис. 35

Вывод данной программы получается такой:

```
изначальная функция  $y = x^2$ 
f(-1,0) = NaN
f(-0,5) = 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(4,5) = NaN
f(5,0) = NaN

работа с точками
копия точки 2: (2.0, 4.0)
после изменения Y точки 2:  $f(1.5) \approx 5.5$ 
после изменения X точки 2:  $f(2.5) = 10.0$ 
после добавления точки (3.2,11):  $f(3.2) = 11.0$ 
после удаления первой точки:  $f(0) = NaN$ 

Process finished with exit code 0
```

Рис. 36

**Спасибо за внимание!**