

## Отчет по лабораторной работе № 2

«Разработка набора классов с табулированными функциями»

По курсу Объектно-ориентированное  
программирование

Выполнил: Акст Роман,  
Студент группы 6203-010302D

## Задание 1

Для работы я использовал IDE IntelliJ IDEA, где создал пакет functions

## Задание 2

Был создан класс FunctionPoint, описывающий точку табулированной функции. Класс содержит два приватных поля для координат x и y, что обеспечивает правильную инкапсуляцию данных. Реализованы три конструктора: конструктор с параметрами координат, конструктор копирования и конструктор по умолчанию. Конструктор копирования создает новый объект с теми же координатами, что и переданная точка, обеспечивая защиту от изменения исходных данных. Геттеры и сеттеры предоставляют контролируемый доступ к полям класса.

```
package functions;

public class FunctionPoint 19 usages  romchik302
{
    private double x; 6 usages
    private double y; 6 usages

    // конструктор по умолчанию
    public FunctionPoint() 20 usages  romchik302
    {
        this.x = 0;
        this.y = 0;
    }

    // конструктор с координатами
    public FunctionPoint(double x, double y) 26 usages  romchik302
    {
        this.x = x;
        this.y = y;
    }

    // конструктор копирования
    public FunctionPoint(FunctionPoint point) 23 usages  romchik302
    {
        this.x = point.x;
        this.y = point.y;
    }

    // геттеры
    public double getX() 19 usages  romchik302
    {
        return x;
    }
}
```

Рисунок 1

```
public double getY() 6 usages  romchik302
{
    return y;
}

// сеттеры
public void setX(double x) no usages  romchik302
{
    this.x = x;
}

public void setY(double y) no usages  romchik302
{
    this.y = y;
}
}
```

Рисунок 2

### Задание 3

Разработан класс `TabulatedFunction` для работы с табулированными функциями. Для хранения точек используется массив объектов `FunctionPoint`, причем точки всегда упорядочены по значению координаты  $x$ . Реализованы два конструктора: первый принимает границы области определения и количество точек, создавая точки с равными интервалами и нулевыми значениями функции; второй конструктор принимает границы и массив значений функции, также равномерно распределяя точки по области определения.

```
package functions;

public class TabulatedFunction 4 usages  romchik302
{
    private FunctionPoint[] points; 38 usages
    private int amountOfElements; 24 usages

    // конструктор для границ координат и количества точек
    public TabulatedFunction(double leftX, double rightX, int pointsCount) 16 usages  romchik302
    {
        this.points = new FunctionPoint[pointsCount];
        this.amountOfElements = pointsCount;

        double step = (rightX - leftX) / (pointsCount - 1);

        for(int i = 0; i < pointsCount; i++)
        {
            this.points[i] = new FunctionPoint(x: leftX + step * i, y: 0);
        }
    }

    // конструктор с граничными значениями X и массивом значений в точках
    public TabulatedFunction(double leftX, double rightX, double[] values) 17 usages  romchik302
    {
        this.points = new FunctionPoint[values.length];
        this.amountOfElements = values.length;

        double step = (rightX - leftX) / (values.length - 1);

        for(int i = 0; i < values.length; i++)
        {
            this.points[i] = new FunctionPoint(x: leftX + step * i, values[i]);
        }
    }
}
```

Рисунок 3

#### Задание 4

В классе `TabulatedFunction` реализованы методы для работы с функцией в целом. Метод `getLeftDomainBorder` возвращает левую границу области определения, соответствующую  $x$  первой точки. Метод `getRightDomainBorder` возвращает правую границу области определения, соответствующую  $x$  последней точки. Метод `getFunctionValue` вычисляет значение функции в произвольной точке  $x$  с использованием линейной интерполяции. Если точка находится вне области определения, метод возвращает `Double.NaN`. Интерполяция реализована через уравнение прямой, проходящей через две соседние точки таблицы. Для избежания погрешности представления была введена переменная `EPSILON`.

```
// геттер левой границы по X
public double getLeftDomainBorder() { return this.points[0].getX(); }

// геттер правой границы по X
public double getRightDomainBorder() no usages 2 romchik302
{
    return (this.amountOfElements == 0) ? Double.NaN : this.points[this.amountOfElements - 1].getX();
}

// получение значения функции по заданной координате
public double getFunctionValue(double x) 1 usage 2 romchik302
{
    if(this.amountOfElements == 0) return Double.NaN;

    // вывод в случае выхода за границы
    if(x < this.points[0].getX() || x > this.points[amountOfElements - 1].getX())
    {
        return Double.NaN;
    }

    // пробегаемся по всем значениям X функции, пока не найдем необходимый
    // либо ищем примерное смежное значение
    for(int i = 0; i < this.amountOfElements - 1; i++)
    {
        double x1 = this.points[i].getX();
        double x2 = this.points[i + 1].getX();

        // сравниваем с машинным эпсилоном для корректного результата
        if(Math.abs(x - x1) < EPSILON) return this.points[i].getY();
        if(Math.abs(x - x2) < EPSILON) return this.points[i + 1].getY();

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

    return Double.NaN;
}
```

Рисунок 4

## Задание 5

Реализован набор методов для работы с отдельными точками функции. Метод `getPointsCount` возвращает количество точек. Метод `getPoint` возвращает копию точки по указанному индексу, что предотвращает несанкционированное изменение исходных данных. Метод `setPoint` заменяет точку на новую, создавая ее копию и проверяя, что новая координата  $x$  не нарушает упорядоченность точек. Методы `getPointX` и `getPointY` возвращают соответствующие координаты точки, а `setPointX` и `setPointY` позволяют изменять координаты с проверкой допустимости изменений.

```
// получение количества точек в функции
public int getPointsCount() { return this.amountOfElements; // возвращаем реальное количество }

// получаем точку по индексу
public FunctionPoint getPoint(int index) no usages 2 romchik302
{
    if(index < 0 || index >= this.amountOfElements)
    {
        return null;
    }

    return new FunctionPoint(this.points[index]);
}

// замена точки по заданному индексу
public void setPoint(int index, FunctionPoint point) 1 usage 2 romchik302
{
    if(index < 0 || index >= this.amountOfElements) return;

    // использование приближенного сравнения для граничных условий
    if (index == 0 && this.amountOfElements > 1 &&
        point.getX() >= points[1].getX() - EPSILON) return;

    if (index == this.amountOfElements - 1 &&
        point.getX() <= points[index - 1].getX() + EPSILON) return;

    if (index > 0 && index < this.amountOfElements - 1 &&
        (point.getX() <= points[index - 1].getX() + EPSILON ||
         point.getX() >= points[index + 1].getX() - EPSILON))
    {
        return;
    }

    points[index] = new FunctionPoint(point);
}
```

Рисунок 5

```

// получение координаты X точки по индексу
public double getPointX(int index)  no usages  👤 romchik302
{
    if(index < 0 || index >= amountOfElements)
    {
        return Double.NaN;
    }

    return this.points[index].getX();
}

// получение координаты Y точки по индексу
public double getPointY(int index)  no usages  👤 romchik302
{
    if(index < 0 || index >= amountOfElements)
    {
        return Double.NaN;
    }

    return this.points[index].getY();
}

// замена значения X в точке с заданным индексом
public void setPointX(int index, double x)  1 usage  👤 romchik302
{
    if(index < 0 || index >= this.amountOfElements) return;

    // проверки из setPoint() адаптированные для setPointX() (для избежания избыточности кода)
    if (index == 0 && this.amountOfElements > 1 && x >= points[1].getX() - EPSILON) return;

    if (index == this.amountOfElements - 1 && x <= points[index - 1].getX() + EPSILON) return;

    if (index > 0 && index < this.amountOfElements - 1 &&
        (x <= points[index - 1].getX() + EPSILON || x >= points[index + 1].getX() - EPSILON))
    {
        return;
    }

    // если все проверки пройдены - меняем X
    this.points[index].setX(x);
}

// замена значения Y в точке с заданным индексом
public void setPointY(int index, double y)  1 usage  👤 romchik302
{
    if(index < 0 || index >= this.amountOfElements) return;

    this.points[index].setY(y);
}

```

Рисунок 6

## Задание 6

Реализованы методы для изменения структуры табулированной функции. Метод `deletePoint` удаляет точку по указанному индексу, сдвигая последующие элементы массива. Метод `addPoint` добавляет новую точку в правильную позицию с учетом упорядоченности по  $x$ . При добавлении точки проверяется необходимость расширения массива, при этом новый массив создается с запасом места для оптимизации последующих операций добавления. Для эффективного копирования данных используется метод `System.arraycopy`.

```
// добавление точки с автоматическим выбором места для подстановки
public void addPoint(FunctionPoint point) 1 usage 2 romchik302
{
    double pointX = point.getX();
    int insertIndex = 0;

    // находим позицию для вставки
    for(; insertIndex < this.amountOfElements; insertIndex++)
    {
        // сравниваем с машинным эпсилоном для избежания погрешности представления
        if(Math.abs(this.points[insertIndex].getX() - pointX) < EPSILON)
        {
            return; // точка с таким X уже существует
        }

        if(this.points[insertIndex].getX() > pointX)
        {
            break;
        }
    }

    // проверяем, нужно ли расширять массив
    if (this.amountOfElements == this.points.length)
    {
        // умножаем на 2 для оптимизации
        FunctionPoint[] newArray = new FunctionPoint[this.points.length * 2 + 1];
        System.arraycopy(this.points, srcPos: 0, newArray, destPos: 0, this.amountOfElements);

        this.points = newArray;
    }

    // сдвигаем элементы вправо для освобождения места
    for (int i = this.amountOfElements; i > insertIndex; i--)
    {
        this.points[i] = this.points[i - 1];
    }

    // вставляем новую точку
    this.points[insertIndex] = new FunctionPoint(point);
    this.amountOfElements++;
}
```

Рисунок 7

## Задание 7

Для проверки корректности работы классов создан класс Main, содержащий точку входа программы. В методе main создается экземпляр TabulatedFunction для линейной функции, тестируются значения функции в различных точках, включая точки внутри и вне области определения. Проверяется работа методов изменения точек, добавления и удаления точек. Результаты выводятся в консоль, также демонстрируется корректность линейной интерполяции и работы всех методов класса.

На рисунках 9 и 10 можем видеть результат выполнения программы для функции  $f(x) = 2x + 1$  на интервале  $[0, 4]$ .

```
// вывод всех точек функции через геттеры
public static void printFunctionPoints(TabulatedFunction func) 5 usages new *
{
    System.out.println("Точки функции:");

    for (int i = 0; i < func.getPointsCount(); i++)
    {
        double x = func.getPointX(i);
        double y = func.getPointY(i);

        System.out.printf(" [%d] (%.2f; %.2f)%n", i, x, y);
    }
}

// метод для вывода координат точки
public static String pointToString(FunctionPoint point) 3 usages new *
{
    if (point == null) return "null";

    return String.format("%.2f; %.2f", point.getX(), point.getY());
}

// проверка работы getFunctionValue
public static void testFunctionValue(TabulatedFunction func) 1 usage 8 romchik302 *
{
    System.out.println("Проверка getFunctionValue (интерполяция):");
    double[] testPoints = {-1, 0, 1.2, 2, 3.5, 4, 5};

    for (double x : testPoints)
    {
        double y = func.getFunctionValue(x);

        if (Double.isNaN(y))
        {
            System.out.printf(" f(%.1f) = не определено%n", x);
        } else
        {
            System.out.printf(" f(%.1f) = %.2f%n", x, y);
        }
    }
}
```

Рисунок 8

```

Исходная функция ( $f(x) = 2x + 1$ ):
Точки функции:
  [0] (0,00; 1,00)
  [1] (1,00; 3,00)
  [2] (2,00; 5,00)
  [3] (3,00; 7,00)
  [4] (4,00; 9,00)
Проверка getFunctionValue (интерполяция):
  f(-1,0) = не определено
  f(0,0) = 1,00
  f(1,2) = 3,40
  f(2,0) = 5,00
  f(3,5) = 8,00
  f(4,0) = 9,00
  f(5,0) = не определено

Заменяем точку с индексом 2:
До: (2,00; 5,00)
После: (2,00; 10,00)
Точки функции:
  [0] (0,00; 1,00)
  [1] (1,00; 3,00)
  [2] (2,00; 10,00)
  [3] (3,00; 7,00)
  [4] (4,00; 9,00)

Добавляем новую точку (1.5, 4):
Точки функции:
  [0] (0,00; 1,00)
  [1] (1,00; 3,00)
  [2] (1,50; 4,00)
  [3] (2,00; 10,00)
  [4] (3,00; 7,00)
  [5] (4,00; 9,00)

```

Рисунок 9

```

Удаляем точку с индексом 1:
Удаляемая точка: (1,00; 3,00)
Точки функции:
  [0] (0,00; 1,00)
  [1] (1,50; 4,00)
  [2] (2,00; 10,00)
  [3] (3,00; 7,00)
  [4] (4,00; 9,00)

Меняем Y точки с индексом 0:
До: Y = 1.0
После: Y = 0.0
Точки функции:
  [0] (0,00; 0,00)
  [1] (1,50; 4,00)
  [2] (2,00; 10,00)
  [3] (3,00; 7,00)
  [4] (4,00; 9,00)

Process finished with exit code 0

```

Рисунок 10