

# Лабораторная 4

Выполнил: Гонеев Роман

Группа: 6201-120303D

Задания:

1. В классах ArrayTabulatedFunction и LinkedListTabulatedFunction добавил конструкторы, получающие сразу все точки функции в виде массива объектов типа FunctionPoint:

- В ArrayTabulatedFunction

```
public ArrayTabulatedFunction(FunctionPoint[] points) {  
    if (points.length < 2) {  
        throw new IllegalArgumentException("Должно быть минимум 2 точки");  
    }  
    // Проверка на сортировку  
    for (int i = 0; i < points.length - 1; ++i) {  
        if (points[i].getX() >= points[i + 1].getX()) {  
            throw new IllegalArgumentException("Абсциссы точек не  
отсортированы");  
        }  
    }  
    // Глубокое копирование для инкапсуляции  
    this.points = new FunctionPoint[points.length];  
    for (int i = 0; i < points.length; ++i) {  
        this.points[i] = new FunctionPoint(points[i]);  
    }  
}
```

- В LinkedListTabulatedFunction

```
public LinkedListTabulatedFunction(FunctionPoint[] points) {  
    if (points.length < 2) {  
        throw new IllegalArgumentException("Должно быть минимум 2 точки");  
    }  
    // Инициализация головы списка  
    this.count = 0;  
    this.head = new FunctionNode();  
    this.head.prev = head;  
    this.head.next = head;  
  
    for (int i = 0; i < points.length; ++i) {  
        // Проверка на сортировку при добавлении  
        if (i > 0 && points[i-1].getX() >= points[i].getX()) {  
            throw new IllegalArgumentException("Абсциссы точек не отсортированы");  
        }  
        // Глубокое копирование  
        addNodeToTail().point = new FunctionPoint(points[i]);  
    }  
}
```

```
        this.count++;
    }
}
```

Оба конструктора проверяют, что в массиве не меньше двух точек, иначе бросают `IllegalArgumentException` и проверяют, что абсциссы точек отсортированы по x по возрастанию, иначе тоже бросают `IllegalArgumentException`.

Копируют точки во внутреннюю структуру, а не хранят ссылку на исходный массив, чтобы сохранить инкапсуляцию.

2. В пакете `functions` создал интерфейс `Function`, описывающий функции одной переменной и содержащий следующие методы:
  - `double getLeftDomainBorder();` - возвращает значение левой границы области определения функции
  - `double getRightDomainBorder();` - возвращает значение правой границы области определения функции;
  - `double getFunctionValue(double x);` - возвращает значение функции в заданной точке.

Исключил соответствующие методы из интерфейса `TabulatedFunction` и сделайте так, чтобы он расширял интерфейс `Function`

```
public interface TabulatedFunction extends Function
```

3. Создал пакет `functions.basic`, в нём будут описаны классы ряда функций, заданных аналитически.  
Создал в пакете публичный класс `Exp`, объекты которого должны вычислять значение экспоненты.

```
package functions.basic;
import functions.Function;

public class Exp implements Function {
    public double getLeftDomainBorder() {
        return Double.NEGATIVE_INFINITY;
    }
    public double getRightDomainBorder() {
        return Double.POSITIVE_INFINITY;
    }
    public double getFunctionValue(double x) {
        return Math.exp(x);
    }
}
```

```
    }  
}
```

Создал класс Log, объекты которого должны вычислять значение логарифма по заданному основанию.

```
package functions.basic;  
import functions.Function;  
  
public class Log implements Function {  
    private double base;  
    public Log(double base) {  
        if (base <= 0 || base == 1) throw new IllegalArgumentException("Некорректное  
основание");  
        this.base = base;  
    }  
    public double getLeftDomainBorder() {  
        return 0;  
    }  
    public double getRightDomainBorder() {  
        return Double.POSITIVE_INFINITY;  
    }  
    public double getFunctionValue(double x) {  
        return Math.log(x) / Math.log(this.base);  
    }  
}
```

Создал класс TrigonometricFunction, реализующий интерфейс Function и описывающий методы получения границ области определения

```
package functions.basic;  
import functions.Function;  
  
public abstract class TrigonometricFunction implements Function {  
  
    public double getLeftDomainBorder() {  
        return Double.NEGATIVE_INFINITY;  
    }  
  
    public double getRightDomainBorder() {  
        return Double.POSITIVE_INFINITY;  
    }  
}
```

Создал наследующие от него публичные классы Sin, Cos и Tan, объекты которых вычисляют, соответственно, значения синуса, косинуса и тангенса:

- package functions.basic;

```
public class Sin extends TrigonometricFunction {  
    public double getFunctionValue(double x) {  
        return Math.sin(x);  
    }  
}
```

- package functions.basic;

```
public class Cos extends TrigonometricFunction {  
    public double getFunctionValue(double x) {  
        return Math.cos(x);  
    }  
}
```

- Tan

```
package functions.basic;
```

```
public class Tan extends TrigonometricFunction {  
    public double getFunctionValue(double x) {  
        return Math.tan(x);  
    }  
}
```

4. Создал пакет functions.meta, в нём будут описаны классы функций, позволяющие комбинировать функции.

Создал класс Sum, объекты которого представляют собой функции, являющиеся суммой двух других функций:

```
package functions.meta;  
import functions.Function;  
  
public class Sum implements Function {  
    private Function f1, f2;  
    private double left, right;  
  
    public Sum(Function f1, Function f2) {  
        this.f1 = f1; this.f2 = f2;
```

```

        this.left = Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
        this.right = Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
    }
    public double getLeftDomainBorder() {
        return left;
    }
    public double getRightDomainBorder() { return right; }
    public double getFunctionValue(double x) { return f1.getFunctionValue(x) +
f2.getFunctionValue(x); }
}

```

Создал класс Mult, объекты которого представляют собой функции, являющиеся произведением двух других функций:

```

package functions.meta;
import functions.Function;

```

```

public class Mult implements Function {
    private Function f1, f2;
    private double left, right;

    public Mult(Function f1, Function f2) {
        this.f1 = f1; this.f2 = f2;
        this.left = Math.max(f1.getLeftDomainBorder(),
f2.getLeftDomainBorder());
        this.right = Math.min(f1.getRightDomainBorder(),
f2.getRightDomainBorder());
    }
    public double getLeftDomainBorder() {
        return left;
    }
    public double getRightDomainBorder() {
        return right;
    }
    public double getFunctionValue(double x) {
        return f1.getFunctionValue(x) * f2.getFunctionValue(x);
    }
}

```

Создал класс Power, объекты которого представляют собой функции, являющиеся степенью другой функции.

```
package functions.meta;
import functions.Function;

public class Power implements Function {
    private Function f;
    private double power;

    public Power(Function f, double power) {
        this.f = f;
        this.power = power;
    }

    public double getLeftDomainBorder() {
        return f.getLeftDomainBorder();
    }
    public double getRightDomainBorder() {
        return f.getRightDomainBorder();
    }
    public double getFunctionValue(double x)
    {
        return Math.pow(f.getFunctionValue(x), power);
    }
}
```

Создал класс Scale, объекты которого описывают функции, полученные из исходных функций путём масштабирования вдоль осей координат:

```
package functions.meta;
import functions.Function;

public class Scale implements Function {
    private Function f;
    private double scaleX, scaleY;

    public Scale(Function f, double scaleX, double scaleY) {
        this.f = f;
```

```
this.scaleX = scaleX;
this.scaleY = scaleY;
}

public double getLeftDomainBorder() {
    return f.getLeftDomainBorder() * scaleX;
}

public double getRightDomainBorder() {
    return f.getRightDomainBorder() * scaleX;
}

public double getFunctionValue(double x) {

    return f.getFunctionValue(x / scaleX) * scaleY;
}
}
```

Создайте класс Shift, объекты которого описывают функции, полученные из исходных функций путём сдвига вдоль осей координат:

```
package functions.meta;
import functions.Function;

public class Shift implements Function {
    private Function f;
    private double shiftX, shiftY;

    public Shift(Function f, double shiftX, double shiftY) {
        this.f = f;
        this.shiftX = shiftX;
        this.shiftY = shiftY;
    }

    public double getLeftDomainBorder() {
        return f.getLeftDomainBorder() + shiftX;
    }
}
```

```
public double getRightDomainBorder() {
    return f.getRightDomainBorder() + shiftX;
}
public double getFunctionValue(double x)
{
    return f.getFunctionValue(x - shiftX) + shiftY;
}
}
```

Создал класс Composition, объекты которого описывают композицию двух исходных функций:

```
package functions.meta;
import functions.Function;

public class Composition implements Function {
    private Function f1, f2;

    public Composition(Function f1, Function f2) {
        this.f1 = f1;
        this.f2 = f2;
    }

    public double getLeftDomainBorder() {
        return f1.getLeftDomainBorder();
    }
    public double getRightDomainBorder() {
        return f1.getRightDomainBorder();
    }
    public double getFunctionValue(double x) {
        // f1(f2(x))
        return f1.getFunctionValue(f2.getFunctionValue(x));
    }
}
```

5. В пакете functions создал класс Functions, содержащий вспомогательные статические методы для работы с функциями и сделал так, чтобы в программе вне этого класса нельзя было создать его объект: `private Functions()`

Класс должен содержать следующие методы:

- `public static Function shift(Function f, double shiftX, double shiftY) {  
 return new Shift(f, shiftX, shiftY);  
}`  
возвращает объект функции, полученной из исходной сдвигом вдоль осей
- `public static Function scale(Function f, double scaleX, double scaleY) {  
 return new Scale(f, scaleX, scaleY);  
}`  
возвращает объект функции, полученной из исходной масштабированием вдоль осей
- `public static Function power(Function f, double power) {  
 return new Power(f, power);  
}`  
возвращает объект функции, являющейся заданной степенью исходной
- `public static Function sum(Function f1, Function f2) {  
 return new Sum(f1, f2);  
}`  
возвращает объект функции, являющейся суммой двух исходных;
- `public static Function mult(Function f1, Function f2) {  
 return new Mult(f1, f2);  
}`  
возвращает объект функции, являющейся произведением двух исходных
- `public static Function composition(Function f1, Function f2) {  
 return new Composition(f1, f2);  
}`  
возвращает объект функции, являющейся композицией двух исходных

6. В пакете functions создал класс TabulatedFunctions, содержащий вспомогательные статические методы для работы с табулированными функциями и сделал так, чтобы в программе вне

этого класса нельзя было создать его объект: `private TabulatedFunctions()`

Описал в классе метод `public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount)`, получающий функцию и возвращающий её табулированный аналог на заданном отрезке с заданным количеством точек:

```
public class TabulatedFunctions {  
    // Приватный конструктор  
    private TabulatedFunctions() {}  
  
    // Метод Задания 6  
    public static TabulatedFunction tabulate(Function function, double leftX,  
        double rightX, int pointsCount) {  
        if (leftX < function.getLeftDomainBorder() || rightX >  
            function.getRightDomainBorder()) {  
            throw new IllegalArgumentException("Границы таблицы за границами  
области определения");  
        }  
        if (pointsCount < 2) {  
            throw new IllegalArgumentException("Должно быть минимум 2  
точки");  
        }  
  
        FunctionPoint[] points = new FunctionPoint[pointsCount];  
        double step = (rightX - leftX) / (pointsCount - 1);  
        for (int i = 0; i < pointsCount; i++) {  
            double x = leftX + i * step;  
            points[i] = new FunctionPoint(x, function.getFunctionValue(x));  
        }  
        // Возвращаем объект одного из классов, реализующих интерфейс  
        return new ArrayTabulatedFunction(points);  
        // Можно вернуть и LinkedListTabulatedFunction  
    }  
}
```

7. В класс `TabulatedFunctions` добавил следующие методы:

- `public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out)` `throws IOException` {  
    `DataOutputStream dos = new DataOutputStream(out);`  
    `dos.writeInt(function.getPointsCount());`  
    `for (int i = 0; i < function.getPointsCount(); i++) {`

```
        dos.writeDouble(function.getPointX(i));
        dos.writeDouble(function.getPointY(i));
    }
    dos.flush(); // Принудительно записываем
}
```

- Метод вывода табулированной функции в байтовый поток

- `public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException {`  
 `DataInputStream dis = new DataInputStream(in);`  
 `int count = dis.readInt();`  
 `FunctionPoint[] points = new FunctionPoint[count];`  
 `for (int i = 0; i < count; i++) {`  
 `points[i] = new FunctionPoint(dis.readDouble(), dis.readDouble());`  
 `}`  
 `return new ArrayTabulatedFunction(points);`  
}

Метод ввода табулированной функции из байтового потока

- `public static void writeTabulatedFunction(TabulatedFunction function, Writer out) throws IOException {`  
 `PrintWriter pw = new PrintWriter(out);`  
 `pw.println(function.getPointsCount());`  
 `for (int i = 0; i < function.getPointsCount(); i++) {`  
 `pw.println(function.getPointX(i) + " " + function.getPointY(i));`  
 `}`  
 `pw.flush();`  
}

Метод записи табулированной функции в символьный поток

- `public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException {`  
 `StreamTokenizer tokenizer = new StreamTokenizer(in);`  
 `tokenizer.nextToken();`  
 `int count = (int) tokenizer.nval;`

```
    FunctionPoint[] points = new FunctionPoint[count];
    for (int i = 0; i < count; i++) {
        tokenizer.nextToken();
        double x = tokenizer.nval;
        tokenizer.nextToken();
        double y = tokenizer.nval;
        points[i] = new FunctionPoint(x, y);
    }
    return new ArrayTabulatedFunction(points);
```

}

Метод чтения табулированной функции из символьного потока

## 8. Проверка работы написанных классов

- Создал по одному объекту классов Sin и Cos и вывел в консоль значения этих функций на отрезке от 0 до  $\pi$  с шагом 0,1:

```
Function sin = new Sin();
Function cos = new Cos();
for (double x = 0; x <= Math.PI; x += 0.1) {
    System.out.printf("x=% .1f, sin(x)=% .4f, cos(x)=% .4f\n", x,
sin.getFunctionValue(x), cos.getFunctionValue(x));
}
```

Вывод:

```
x=0,0, sin(x)=0,0000, cos(x)=1,0000
x=0,1, sin(x)=0,0998, cos(x)=0,9950
x=0,2, sin(x)=0,1987, cos(x)=0,9801
x=0,3, sin(x)=0,2955, cos(x)=0,9553
x=0,4, sin(x)=0,3894, cos(x)=0,9211
x=0,5, sin(x)=0,4794, cos(x)=0,8776
x=0,6, sin(x)=0,5646, cos(x)=0,8253
x=0,7, sin(x)=0,6442, cos(x)=0,7648
x=0,8, sin(x)=0,7174, cos(x)=0,6967
x=0,9, sin(x)=0,7833, cos(x)=0,6216
x=1,0, sin(x)=0,8415, cos(x)=0,5403
x=1,1, sin(x)=0,8912, cos(x)=0,4536
x=1,2, sin(x)=0,9320, cos(x)=0,3624
x=1,3, sin(x)=0,9636, cos(x)=0,2675
x=1,4, sin(x)=0,9854, cos(x)=0,1700
x=1,5, sin(x)=0,9975, cos(x)=0,0707
x=1,6, sin(x)=0,9996, cos(x)=-0,0292
x=1,7, sin(x)=0,9917, cos(x)=-0,1288
x=1,8, sin(x)=0,9738, cos(x)=-0,2272
x=1,9, sin(x)=0,9463, cos(x)=-0,3233
x=2,0, sin(x)=0,9093, cos(x)=-0,4161
x=2,1, sin(x)=0,8632, cos(x)=-0,5048
x=2,2, sin(x)=0,8085, cos(x)=-0,5885
x=2,3, sin(x)=0,7457, cos(x)=-0,6663
x=2,4, sin(x)=0,6755, cos(x)=-0,7374
x=2,5, sin(x)=0,5985, cos(x)=-0,8011
x=2,6, sin(x)=0,5155, cos(x)=-0,8569
x=2,7, sin(x)=0,4274, cos(x)=-0,9041
x=2,8, sin(x)=0,3350, cos(x)=-0,9422
```

```
x=2,9, sin(x)=0,2392, cos(x)=-0,9710  
x=3,0, sin(x)=0,1411, cos(x)=-0,9900  
x=3,1, sin(x)=0,0416, cos(x)=-0,9991
```

- С помощью метода TabulatedFunctions.tabulate() создал табулированные аналоги этих функций на отрезке от 0 до  $\pi$  с 10 точками и вывел в консоль значения этих функций на отрезке от 0 до  $\pi$  с шагом 0,1:

```
System.out.println("\nЗадание 8.2: Табулирование sin и cos, 5 точек ");  
TabulatedFunction sintab = TabulatedFunctions.tabulate(sin, 0, Math.PI, 10);  
TabulatedFunction costab = TabulatedFunctions.tabulate(cos, 0, Math.PI, 10);  
for (double x = 0; x <= Math.PI; x += 0.1) {  
    System.out.printf("x=% .1f, sintab(x)=% .4f, costab(x)=% .4f\n", x,  
        sintab.getFunctionValue(x), costab.getFunctionValue(x));  
}  
  
x=0,0, tabSin(x)=0,0000, tabCos(x)=1,0000  
x=0,1, tabSin(x)=0,0980, tabCos(x)=0,9827  
x=0,2, tabSin(x)=0,1960, tabCos(x)=0,9654  
x=0,3, tabSin(x)=0,2939, tabCos(x)=0,9482  
x=0,4, tabSin(x)=0,3859, tabCos(x)=0,9144  
x=0,5, tabSin(x)=0,4721, tabCos(x)=0,8646  
x=0,6, tabSin(x)=0,5582, tabCos(x)=0,8149  
x=0,7, tabSin(x)=0,6440, tabCos(x)=0,7646  
x=0,8, tabSin(x)=0,7079, tabCos(x)=0,6884  
x=0,9, tabSin(x)=0,7719, tabCos(x)=0,6122  
x=1,0, tabSin(x)=0,8358, tabCos(x)=0,5360  
x=1,1, tabSin(x)=0,8840, tabCos(x)=0,4506  
x=1,2, tabSin(x)=0,9180, tabCos(x)=0,3571  
x=1,3, tabSin(x)=0,9521, tabCos(x)=0,2636  
x=1,4, tabSin(x)=0,9848, tabCos(x)=0,1699  
x=1,5, tabSin(x)=0,9848, tabCos(x)=0,0704  
x=1,6, tabSin(x)=0,9848, tabCos(x)=-0,0291  
x=1,7, tabSin(x)=0,9848, tabCos(x)=-0,1285  
x=1,8, tabSin(x)=0,9662, tabCos(x)=-0,2248  
x=1,9, tabSin(x)=0,9322, tabCos(x)=-0,3183  
x=2,0, tabSin(x)=0,8981, tabCos(x)=-0,4117  
x=2,1, tabSin(x)=0,8624, tabCos(x)=-0,5043  
x=2,2, tabSin(x)=0,7985, tabCos(x)=-0,5805  
x=2,3, tabSin(x)=0,7345, tabCos(x)=-0,6567  
x=2,4, tabSin(x)=0,6706, tabCos(x)=-0,7329  
x=2,5, tabSin(x)=0,5941, tabCos(x)=-0,7942  
x=2,6, tabSin(x)=0,5079, tabCos(x)=-0,8439
```

```
x=2,7, tabSin(x)=0,4217, tabCos(x)=-0,8937  
x=2,8, tabSin(x)=0,3347, tabCos(x)=-0,9410  
x=2,9, tabSin(x)=0,2367, tabCos(x)=-0,9583  
x=3,0, tabSin(x)=0,1387, tabCos(x)=-0,9755  
x=3,1, tabSin(x)=0,0408, tabCos(x)=-0,9928
```

С помощью методов класса Functions создал объект функции, являющейся суммой квадратов табулированных аналогов синуса и косинуса и вывел в консоль значения этой функций на отрезке от 0 до  $\pi$  с шагом 0,1:

```
System.out.println("\nRj Задание 8.3: Сумма квадратов: sin^2 + cos^2");  
Function sinSqr = Functions.power(sintab, 2);  
Function cosSqr = Functions.power(costab, 2);  
Function sumSqr = Functions.sum(sinSqr, cosSqr);  
for (double x = 0; x <= Math.PI; x += 0.1) {  
    System.out.printf("x=% .1f, sin^2+cos^2=% .4f\n", x,  
        sumSqr.getFunctionValue(x));  
}
```

Вывод:

```
x=0,0, sin^2+cos^2=1,0000  
x=0,1, sin^2+cos^2=0,9753  
x=0,2, sin^2+cos^2=0,9705  
x=0,3, sin^2+cos^2=0,9854  
x=0,4, sin^2+cos^2=0,9850  
x=0,5, sin^2+cos^2=0,9704  
x=0,6, sin^2+cos^2=0,9756  
x=0,7, sin^2+cos^2=0,9994  
x=0,8, sin^2+cos^2=0,9751  
x=0,9, sin^2+cos^2=0,9706  
x=1,0, sin^2+cos^2=0,9859  
x=1,1, sin^2+cos^2=0,9845  
x=1,2, sin^2+cos^2=0,9703  
x=1,3, sin^2+cos^2=0,9759  
x=1,4, sin^2+cos^2=0,9987  
x=1,5, sin^2+cos^2=0,9748  
x=1,6, sin^2+cos^2=0,9707  
x=1,7, sin^2+cos^2=0,9864  
x=1,8, sin^2+cos^2=0,9841  
x=1,9, sin^2+cos^2=0,9702  
x=2,0, sin^2+cos^2=0,9762  
x=2,1, sin^2+cos^2=0,9981
```

```
x=2,2, sin^2+cos^2=0,9745
x=2,3, sin^2+cos^2=0,9708
x=2,4, sin^2+cos^2=0,9869
x=2,5, sin^2+cos^2=0,9836
x=2,6, sin^2+cos^2=0,9702
x=2,7, sin^2+cos^2=0,9765
x=2,8, sin^2+cos^2=0,9975
x=2,9, sin^2+cos^2=0,9743
x=3,0, sin^2+cos^2=0,9709
x=3,1, sin^2+cos^2=0,9873
```

- С помощью метода `TabulatedFunctions.tabulate()` создал табулированный аналог логарифма по натуральному основанию на отрезке от 0 до 10 с 11 точками:

```
System.out.println("\nЗадание 8.5: Тест output/input");
try {
    TabulatedFunction tabLog = TabulatedFunctions.tabulate(new
        Log(Math.E), 0, 10, 10);

    // Запись в файл
    OutputStream out = new FileOutputStream("Табулированный-log.dat");
    TabulatedFunctions.outputTabulatedFunction(tabLog, out);
    out.close();

    // Чтение из файла
    InputStream in = new FileInputStream("Табулированный-log.dat");
    TabulatedFunction readLog =
        TabulatedFunctions.inputTabulatedFunction(in);
    in.close();
```

Вывел и сравнил значения исходной и считанной функции на отрезке от 0 до 10 с шагом 1:

```
System.out.println("Сравнение исходной и считанной с файла функции:");
for (double x = 1; x <= 10; x += 1) {
    System.out.printf("x=%,.2f, Orig(x)=%,.6f, Read(x)=%,.6f\n", x,
        tabLog.getFunctionValue(x), readLog.getFunctionValue(x));
}
```

```
x=1,0, Orig(x)=0,00, Read(x)=0,00
x=2,0, Orig(x)=0,69, Read(x)=0,69
x=3,0, Orig(x)=1,10, Read(x)=1,10
x=4,0, Orig(x)=1,39, Read(x)=1,39
```

```
x=5,0, Orig(x)=1,61, Read(x)=1,61  
x=6,0, Orig(x)=1,79, Read(x)=1,79  
x=7,0, Orig(x)=1,95, Read(x)=1,95  
x=8,0, Orig(x)=2,08, Read(x)=2,08  
x=9,0, Orig(x)=2,20, Read(x)=2,20  
x=10,0, Orig(x)=2,30, Read(x)=2,30
```

9. Сделал так, чтобы объекты всех классов, реализующих интерфейс TabulatedFunction, были сериализуемыми:  
ArrayTabulatedFunction implements TabulatedFunction,  
SerializableLinkedListTabulatedFunction implements TabulatedFunction,  
Externalizable.

```
System.out.println("\nЗадание 9: Тест Сериализации - Экстерьерализация,  
LinkedListTabulatedFunction");  
try {  
    // Создаем функцию f(x) = x^2  
    Function sqr = new Power(new IdentityFunction(), 2);  
  
    // Создаем табулированную функцию на основе  
    LinkedListTabulatedFunction  
    double[] values = {0, 1, 4, 9, 16};  
    TabulatedFunction tabFuncLinked = new LinkedListTabulatedFunction(0, 4,  
values);  
  
    System.out.println("Исходная функция (LinkedList):");  
    for (int i = 0; i < tabFuncLinked.getPointsCount(); i++) {  
        System.out.printf("Point %d: (%.1f, %.1f)\n", i, tabFuncLinked.getPointX(i),  
tabFuncLinked.getPointY(i));  
    }
```

Сериализовал полученный объект в файл:

```
ObjectOutputStream out = new ObjectOutputStream(new  
FileOutputStream("Сериализованный-linkedlist.ser"));  
out.writeObject(tabFuncLinked);  
out.close();
```

Далее десериализовал табулированную функцию из этого файла:

```
ObjectInputStream in = new ObjectInputStream(new  
FileInputStream("Сериализованный-linkedlist.ser"));  
TabulatedFunction readFuncLinked = (TabulatedFunction) in.readObject();  
in.close();
```

Вывод:

x=0,0, Orig(x)=0,00, Read(x)=0,00  
x=1,0, Orig(x)=1,00, Read(x)=1,00  
x=2,0, Orig(x)=2,00, Read(x)=2,00  
x=3,0, Orig(x)=3,00, Read(x)=3,00  
x=4,0, Orig(x)=4,00, Read(x)=4,00  
x=5,0, Orig(x)=5,00, Read(x)=5,00  
x=6,0, Orig(x)=6,00, Read(x)=6,00  
x=7,0, Orig(x)=7,00, Read(x)=7,00  
x=8,0, Orig(x)=8,00, Read(x)=8,00  
x=9,0, Orig(x)=9,00, Read(x)=9,00  
x=10,0, Orig(x)=10,00, Read(x)=10,00

Исходная функция (LinkedList):

Point 0: (0,0, 0,0)  
Point 1: (1,0, 1,0)  
Point 2: (2,0, 4,0)  
Point 3: (3,0, 9,0)  
Point 4: (4,0, 16,0)

Сравнение исходной и десериализованной функции (LinkedList):

x=0,0, Orig(x)=0,00, Read(x)=0,00  
x=0,5, Orig(x)=0,50, Read(x)=0,50  
x=1,0, Orig(x)=1,00, Read(x)=1,00  
x=1,5, Orig(x)=2,50, Read(x)=2,50  
x=2,0, Orig(x)=4,00, Read(x)=4,00  
x=2,5, Orig(x)=6,50, Read(x)=6,50  
x=3,0, Orig(x)=9,00, Read(x)=9,00  
x=3,5, Orig(x)=12,50, Read(x)=12,50  
x=4,0, Orig(x)=16,00, Read(x)=16,00