

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное  
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет  
имени академика С.П. Королева»  
(Самарский университет)

ОТЧЕТ ПО  
ЛАБОРАТОРНОЙ РАБОТЕ № 4

**«Создание классов математических  
функций, файловый ввод-вывод, работа  
с сериализацией»**

по курсу  
Объектно-ориентированное программирование

Выполнил: Петрухин Роман,  
студент группы 6203-010302D

## Оглавление

Пункт №1 .....	3
Пункт №2 .....	3-4
Пункт №3 .....	5-8
Пункт №4 .....	8
Пункт №5 .....	9-11
Пункт №6 .....	11-14

## Пункт №1

Реализация данного задания представлена на рисунках 1,2.

```
public ArrayTabulatedFunction(FunctionPoint[] massiveOfPoints) throws IllegalArgumentException,
    this.amountOfElements = massiveOfPoints.length;
    if (amountOfElements < 2)
        throw new IllegalArgumentException("Massive length must be greater than 2!");

    for (int i = 1; i < amountOfElements; i++) {
        if (massiveOfPoints[i].getX() < massiveOfPoints[i-1].getX()) {
            throw new InappropriateFunctionPointException();
        }
    }

    this.massiveOfPoints = new FunctionPoint[amountOfElements];
    for (int i = 0; i < amountOfElements; i++) {
        this.massiveOfPoints[i] = new FunctionPoint(massiveOfPoints[i]);
    }
}
```

Рисунок 1

```
public LinkedListTabulatedFunction(FunctionPoint[] points) 2 usages &zhestok1
    throws IllegalArgumentException, InappropriateFunctionPointException {
    if (points == null) throw new IllegalArgumentException("Points array cannot be null");
    if (points.length < 2) throw new IllegalArgumentException("Massive must be greater than 2!");

    for (int i = 1; i < points.length; i++) {
        if (points[i].getX() <= points[i-1].getX()) {
            throw new InappropriateFunctionPointException();
        }
    }

    for (int i = 0; i < points.length; i++) {
        addNodeToTail(new FunctionPoint(points[i]));
    }
}
```

Рисунок 2

## Пункт №2

Реализация данного задания представлена на рисунке 3.

```

public interface Function {  @zhestok1
    /**
     * Возвращает левую границу области определения функции
     * @return значение левой границы области определения
     */
    double getLeftDomainBorder(); 11 implementations  @zhestok1

    /**
     * Возвращает правую границу области определения функции
     * @return значение правой границы области определения
     */
    double getRightDomainBorder(); 11 implementations  @zhestok1

    /**
     * Вычисляет значение функции в заданной точке с помощью линейной интерполяции
     * @param x координата, в которой вычисляется значение функции
     * @return значение функции в точке x или Double.NaN, если x вне области определения
     */
    double getFunctionValue(double x); 14 implementations  @zhestok1
}

```

Рисунок 3

## Пункт №3

Реализация данного задания представлена на рисунках 4 -7. Все спорные моменты написаны в качестве комментариев рядом с функциями.

```

package functions.basic;

import functions.Function;

public abstract class TrigonometricFunction implements Function {

    @Override
    public double getLeftDomainBorder() {
        return Double.NEGATIVE_INFINITY;
    }

    @Override
    public double getRightDomainBorder() {
        return Double.POSITIVE_INFINITY;
    }

    @Override
    public abstract double getFunctionValue(double x);
}

```

Рисунок 4

```

package functions.basic;

public class Tan extends TrigonometricFunction {

    @Override
    public double getFunctionValue(double x) {
        return Math.tan(x);
    }
}

```

Рисунок 5

Реализация классов косинуса и синуса выглядит ровно также как и тангенса, поэтому показывать их не вижу никакого смысла. Поэтому далее будут представлены логарифм и экспонента.

```

public class Log implements Function { 3 usages  zhestok1

    private double logBase; 2 usages

    /**
     * Конструктор класса Log
     * @param logBase основание логарифма
     * @throws IllegalArgumentException некорректное основание логарифма
     */
    public Log(double logBase) throws IllegalArgumentException { 5 usages  zhestok1
        if (logBase <= 0 || logBase == 1) {
            throw new IllegalArgumentException("Incorrect base of log!");
        }
        this.logBase = logBase;
    }

    @Override  zhestok1
    // Логарифм определен для x > 0
    public double getLeftDomainBorder() {
        return 0;
    }

    @Override  zhestok1
    // Логарифм устремляется к бесконечности
    public double getRightDomainBorder() {
        return Double.POSITIVE_INFINITY;
    }

    @Override  zhestok1
    public double getFunctionValue(double x) {
        if (x < 0) return Double.NaN;
        return Math.log(x) / Math.log(logBase);
    }
}

```

Рисунок 6

```

public class Exp implements Function { 3 usages  zhestok1

    @Override zhestok1
    // Если хотим использовать левую границу этого типа, то подойдет и MIN_EXPONENT()
    // Но более математически точным будет возврат NEGATIVE_INFINITY
    public double getLeftDomainBorder() {
        return Double.NEGATIVE_INFINITY;
    }

    @Override zhestok1
    // Если хотим использовать правую границу этого типа, то подойдет и MAX_EXPONENT()
    // Но более математически точным будет возврат POSITIVE_INFINITY
    public double getRightDomainBorder() {
        return Double.POSITIVE_INFINITY;
    }

    @Override zhestok1
    public double getFunctionValue(double x) {
        return Math.exp(x);
    }
}

```

Рисунок 7

## Пункт №4

Все классы данного пакета (meta) имеют одинаковую структуру, т.к. имплементируют интерфейс Function. Конструкторы этих классов выглядят тоже абсолютно одинаково, поэтому показывать их не вижу никакого смысла. А реализация всех функций из этого задания представлена на рисунках 8 - 13.

```

@Override @zhestok1
public double getLeftDomainBorder() {

    double leftBorder1 = firstFunc.getLeftDomainBorder();
    double leftBorder2 = secondFunc.getLeftDomainBorder();

    return Math.max(leftBorder1, leftBorder2);
}

@Override @zhestok1
public double getRightDomainBorder() {

    double rightBorder1 = firstFunc.getRightDomainBorder();
    double rightBorder2 = firstFunc.getRightDomainBorder();

    return Math.min(rightBorder1, rightBorder2);
}

@Override @zhestok1
public double getFunctionValue(double x) {
    // Проверяем, что x принадлежит пересечению областей определения
    double leftBorder = getLeftDomainBorder();
    double rightBorder = getRightDomainBorder();

    if (x < leftBorder || x > rightBorder) {
        return Double.NaN;
    }

    // Суммируем значения функций
    return firstFunc.getFunctionValue(x) + secondFunc.getFunctionValue(x);
}
}

```

Рисунок 8. Класс Sum

```

@Override @zhestok1
public double getLeftDomainBorder() {
    return func.getLeftDomainBorder() - xCoefficient; // Если +, то влево, если минус то вправо
}

@Override @zhestok1
public double getRightDomainBorder() {
    return func.getRightDomainBorder() - xCoefficient;
}

@Override @zhestok1
public double getFunctionValue(double x) {
    return func.getFunctionValue(x: x + xCoefficient) + yCoefficient; // Сдвиг по y
}
}

```

Рисунок 9. Класс Shift



```

@Override @zhestok1
public double getLeftDomainBorder() {
    if (xCoefficient > 0) {
        return func.getLeftDomainBorder() / xCoefficient;
    } else if (xCoefficient < 0) {
        return func.getRightDomainBorder() / xCoefficient;
    } else {
        // scaleX = 0 - функция определена везде
        return Double.NEGATIVE_INFINITY;
    }
}

@Override @zhestok1
public double getRightDomainBorder() {
    if (xCoefficient > 0) {
        return func.getRightDomainBorder() / xCoefficient;
    } else if (xCoefficient < 0) {
        return func.getLeftDomainBorder() / xCoefficient;
    } else {
        // scaleX = 0 - функция определена везде
        return Double.POSITIVE_INFINITY;
    }
}

@Override @zhestok1
public double getFunctionValue(double x) {
    // Проверяем, что x принадлежит области определения
    double leftBorder = getLeftDomainBorder();
    double rightBorder = getRightDomainBorder();

    if (x < leftBorder || x > rightBorder) {
        return Double.NaN;
    }

    // Масштабируем аргумент и результат
    return func.getFunctionValue(x * x * xCoefficient) * yCoefficient; // Умножаю на xCoefficient потому что раньше делил
}

```

Рисунок 10. Класс Scale

```

@Override @zhestok1
public double getLeftDomainBorder() {
    return funcBase.getLeftDomainBorder();
}

@Override @zhestok1
public double getRightDomainBorder() {
    return funcBase.getRightDomainBorder();
}

@Override @zhestok1
public double getFunctionValue(double x) {
    return Math.pow(funcBase.getFunctionValue(x), power);
}

```

Рисунок 11. Класс Power

```

@Override @zhestok1
public double getRightDomainBorder() {

    double rightBorder1 = firstFunc.getRightDomainBorder();
    double rightBorder2 = firstFunc.getRightDomainBorder();

    return Math.min(rightBorder1, rightBorder2);
}

@Override @zhestok1
public double getFunctionValue(double x) {
    // Проверяем, что x принадлежит пересечению областей определения
    double leftBorder = getLeftDomainBorder();
    double rightBorder = getRightDomainBorder();

    if (x < leftBorder || x > rightBorder) {
        return Double.NaN;
    }

    // Умножаем значения функций
    return firstFunc.getFunctionValue(x) * secondFunc.getFunctionValue(x);
}

```

Рисунок 12. Класс Mult

```

@Override @zhestok1
public double getLeftDomainBorder() {
    return secondFunc.getLeftDomainBorder();
}

@Override @zhestok1
public double getRightDomainBorder() {
    return firstFunc.getRightDomainBorder(); // Могу взять разные так как области совпадают
}

@Override @zhestok1
public double getFunctionValue(double x) {

    double secondVal = secondFunc.getFunctionValue(x);
    return firstFunc.getFunctionValue(secondVal);
}

```

Рисунок 13. Класс Composition

## Пункт №5

В задании №5 я должен создать класс, который отвечает за создание объектов. Конструктор я сделал недоступным для создания объектов, сделав его private. Реализация представлена на рисунках 14, 15.

```

private Functions() {} no usages  zhestok1

/**
 * Статический метод создания объекта типа Shift
 * @param f функция
 * @param shiftX сдвиг по x
 * @param shiftY сдвиг по y
 * @return объект
 */
public static Function shift(Function f, double shiftX, double shiftY) { no usages  zhestok1
    return new Shift(f, shiftX, shiftY);
}

/**
 * Статический метод создания объекта типа Scale
 * @param f функция
 * @param scaleX сдвиг по x
 * @param scaleY сдвиг по y
 * @return объект
 */
public static Function scale(Function f, double scaleX, double scaleY) { no usages  zhestok1
    return new Scale(f, scaleX, scaleY);
}

```

Рисунок 14

```

public static Function power(Function f, double power) { no usages  zhestok1
    return new Power(f, power);
}

/**
 * Статический метод создания объекта типа Sum
 * @param f1 функция 1
 * @param f2 функция 2
 * @return объект
 */
public static Function sum(Function f1, Function f2) { no usages  zhestok1
    return new Sum(f1, f2);
}

/**
 * Статический метод создания объекта типа Mult
 * @param f1 функция 1
 * @param f2 функция 2
 * @return объект
 */
public static Function mult(Function f1, Function f2) { no usages  zhestok1
    return new Mult(f1, f2);
}

/**
 * Статический метод создания объекта типа Composition
 * @param f1
 * @param f2
 * @return объект
 */
public static Function composition(Function f1, Function f2) { no usages  zhestok1
    return new Composition(f1, f2);
}

```

Рисунок 15

## Пункт №6

В 6 таске нужно реализовать функцию `tabulate()`. И опять приватный конструктор. Конструктор был показан выше. А реализация `tabulate` представлена на рисунке 16.

```

public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
    // Проверка входных параметров
    if (function == null) {
        throw new IllegalArgumentException("Function cannot be null");
    }
    if (leftX >= rightX) {
        throw new IllegalArgumentException("LeftX cannot be greater or equal to rightX");
    }
    if (pointsCount < 2) {
        throw new IllegalArgumentException("Points count must be greater than 1");
    }

    // Проверка, что отрезок табулирования принадлежит области определения
    if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
        throw new IllegalArgumentException("Tabulation interval is outside function domain");
    }

    // Создание табулированной функции
    TabulatedFunction tabulatedFunc = new ArrayTabulatedFunction(leftX, rightX, pointsCount);

    // Заполнение значений функции
    for (int i = 0; i < pointsCount; i++) {
        double x = tabulatedFunc.getPointX(i);
        double y = function.getFunctionValue(x);
        tabulatedFunc.setPointY(i, y);
    }

    return tabulatedFunc;
}

```

Рисунок 16

## Пункт №7

Пункт посвящен чтению и записи данных в файл и из него соответственно. В качестве обёрток для первых трёх функций я использовал `DataOutputStream`, `DataInputStream` и `BufferedWriter` соответственно. Функцию `readTabulatedFunction` как и написано в задании я сделал при помощи `StreamTokenizer`. Реализация представлена на рисунке 17-20.

```

public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) 1 usage
throws IOException {

    // Заккрытие потоков это ответственность вызывающего кода
    DataOutputStream data = new DataOutputStream(out); // DataOutputStream - поток обёртка
    try {
        int pCount = function.getPointsCount();
        data.writeInt(pCount);

        for (int i = 0; i < pCount; i++) {
            double x = function.getPointX(i);
            double y = function.getPointY(i);
            data.writeDouble(x);
            data.writeDouble(y);
        }

        data.flush(); // сбрасываем буфер
    } catch (IOException e) {
        throw new IOException("Have a problem with your flow!", e); // Пробрасываем исключение
    }
}

```

Рисунок 17

```

public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException, InappropriateFunctionPointException
{
    DataInputStream data = new DataInputStream(in);

    FunctionPoint[] points;
    try {
        int pCount = data.readInt();
        points = new FunctionPoint[pCount];
        for (int i = 0; i < pCount; i++) {
            points[i] = new FunctionPoint(data.readDouble(), data.readDouble());
        }
    } catch (IOException e) {
        throw new IOException("Have a problem with your flow!", e);
    }

    return new ArrayTabulatedFunction(points);
}

```

Рисунок 18

```

public static void writeTabulatedFunction(TabulatedFunction function, Writer out) 1 usage 2 zhestok1
    throws IOException {

    BufferedWriter bfWriter = new BufferedWriter(out);

    try {
        int pCount = function.getPointsCount();

        // ПРАВИЛЬНАЯ запись числа как строки
        bfWriter.write(String.valueOf(pCount));
        bfWriter.write(str: " ");

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

            // Записываем каждое значение отдельно с пробелами
            bfWriter.write(String.valueOf(x));
            bfWriter.write(str: " ");
            bfWriter.write(String.valueOf(y));

            // Добавляем пробел между точками (кроме последней)
            if (i < pCount - 1) {
                bfWriter.write(str: " ");
            }
        }

        bfWriter.flush();

    } catch (IOException e) {
        throw new IOException("Error writing tabulated function to writer", e);
    }
}

```

Рисунок 19

```

public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException, InappropriateFunctionPointException {
    StreamTokenizer tokenizer = new StreamTokenizer(in);

    try {
        if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) { // TT_NUMBER - токен является числом
            throw new IOException("Expected points count number");
        }
        int pointsCount = (int) tokenizer.nval;

        FunctionPoint[] points = new FunctionPoint[pointsCount];
        for (int i = 0; i < pointsCount; i++) {
            // Читаем x
            if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
                throw new IOException("Expected x coordinate at point " + i);
            }
            double x = tokenizer.nval;

            // Читаем y
            if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
                throw new IOException("Expected y coordinate at point " + i);
            }
            double y = tokenizer.nval;

            points[i] = new FunctionPoint(x, y);
        }

        return new ArrayTabulatedFunction(points);
    } catch (IOException e) {
        throw new IOException("Have a problem with your flow!", e);
    } catch (InappropriateFunctionPointException e) {
        throw new RuntimeException(e);
    }
}

```

Рисунок 20

По поводу вопросов об обработке `IOException`. Я почитал статейки от умных людей на JavaRush и Hubr и решил, что обработка ошибки `IO` необходима, потому автор кода обязан знать об этих ошибках, ибо некорректные данные в потоке могут привести к неожиданным результатам. А закрытие потоков это прерогатива того, кто использует этот функционал, то есть человек который использует твои объекты обязан самостоятельно следить за открытием и закрытием этих самых потоков.

## Пункт №8

Main, результат её работы и содержимое файлов будет продемонстрировано на рисунках далее.

```
public class Main { @zhestok1
    public static void main(String[] args) throws InappropriateFunctionPointException, IOException { @zhestok1

        Function sin = new Sin();
        Function cos = new Cos();

        double from = 0;
        double to = Math.PI;
        double step = 0.1;

        System.out.println("Sin values:");
        for (double x = from; x <= to + 1e-10; x += step) {
            System.out.printf("sin(%.1f) = %.6f\n", x, sin.getFunctionValue(x));
        }

        System.out.println("\nCos values:");
        for (double x = from; x <= to + 1e-10; x += step) {
            System.out.printf("cos(%.1f) = %.6f\n", x, cos.getFunctionValue(x));
        }

        System.out.println("\n== Табулированные аналоги ==");
        int pointsCount = 10;
        TabulatedFunction tabulatedSin = TabulatedFunctions.tabulate(sin, from, to, pointsCount);
        TabulatedFunction tabulatedCos = TabulatedFunctions.tabulate(cos, from, to, pointsCount);

        System.out.println("Табулированный Sin:");
        for (double x = from; x <= to + 1e-10; x += step) {
            double original = sin.getFunctionValue(x);
            double tabulated = tabulatedSin.getFunctionValue(x);
            System.out.printf("x=%.1f: original=%.6f, tabulated=%.6f, diff=%.6f\n",
                x, original, tabulated, Math.abs(original - tabulated));
        }

        System.out.println("\n== Сумма квадратов ==");
    }
}
```

Рисунок 21



```

// Квадраты функций
Power sinSquared = new Power(tabulatedSin, power: 2);
Power cosSquared = new Power(tabulatedCos, power: 2);

// Сумма квадратов
Sum sumOfSquares = new Sum(sinSquared, cosSquared);

System.out.println("Сумма квадратов с " + pointsCount + " разными точками:");
for (double x = from; x <= to + 1e-10; x += step) {
    System.out.printf("x=%.1f: result=%.6f\n", x, sumOfSquares.getFunctionValue(x));
}

// Исследуем влияние количества точек
System.out.println("\n=== Влияние количества точек ===");
int[] pointCounts = {5, 10, 20, 50};
for (int count : pointCounts) {
    TabulatedFunction sinTab = TabulatedFunctions.tabulate(sin, from, to, count);
    TabulatedFunction cosTab = TabulatedFunctions.tabulate(cos, from, to, count);
    Sum sum = new Sum(new Power(sinTab, power: 2), new Power(cosTab, power: 2));

    double error = 0;
    for (double x = from; x <= to + 1e-10; x += step) {
        double actual = sum.getFunctionValue(x);
        double expected = 1.0; //  $\sin^2 + \cos^2 = 1$ 
        error += Math.abs(actual - expected);
    }
    System.out.printf("Точек: %d, средняя ошибка: %.6f\n", count, error/((to-from)/step + 1));
}

```

Рисунок 22

```

// 4. Работа с экспонентой (бинарный формат)
System.out.println("\n=== Экспонента (бинарный формат) ===");
Function exp = new Exp();
TabulatedFunction tabulatedExp = TabulatedFunctions.tabulate(exp, leftX: 0, rightX: 10, pointsCount: 11);

// Запись в файл
try (FileOutputStream fos = new FileOutputStream("exp.txt")) {
    TabulatedFunctions.outputTabulatedFunction(tabulatedExp, fos);
}

// Чтение из файла
TabulatedFunction readExp;
try (FileInputStream fis = new FileInputStream("exp.txt")) {
    readExp = TabulatedFunctions.inputTabulatedFunction(fis);
}

System.out.println("Сравнение экспоненты:");
for (double x = 0; x <= 10; x += 1) {
    double original = tabulatedExp.getFunctionValue(x);
    double fromFile = readExp.getFunctionValue(x);
    System.out.printf("x=%.0f: original=%.6f, fromFile=%.6f, equal=%b\n",
        x, original, fromFile, Math.abs(original - fromFile) < 1e-10);
}

```

Рисунок 23

```
// 5. Работа с логарифмом (текстовый формат)
System.out.println("\n=== Логарифм (текстовый формат) ===");
Function ln = new Log(Math.E);
TabulatedFunction tabulatedLn = TabulatedFunctions.tabulate(ln, leftX: 1, rightX: 10, pointsCount: 10); // от 1 т.к. ln(0) не определен

// Запись в файл
try (FileWriter fw = new FileWriter( fileName: "ln.txt")) {
    TabulatedFunctions.writeTabulatedFunction(tabulatedLn, fw);
}

// Чтение из файла
TabulatedFunction readLn;
try (FileReader fr = new FileReader( fileName: "ln.txt")) {
    readLn = TabulatedFunctions.readTabulatedFunction(fr);
}

System.out.println("Сравнение логарифма:");
for (double x = 1; x <= 10; x += 1) {
    double original = tabulatedLn.getFunctionValue(x);
    double fromFile = readLn.getFunctionValue(x);
    System.out.printf("x=%0f: original=%0.6f, fromFile=%0.6f, equal=%b\n",
        x, original, fromFile, Math.abs(original - fromFile) < 1e-10);
}

// Бинарный формат
// + Меньший размер файла
// + Точное представление чисел
// + Быстрая запись/чтение
// - Не читаем для человека

// Текстовый формат
// + Читаемость для человека
// + Легко редактировать
// - Большой размер файла
```

Рисунок 24

Продемонстрирую лишь часть результатов дабы слишком не растягивать отчёт. Не хочу вставлять скриншоты длиной в страницу только с результатами работы функции sin и cos.

Sin values:	Cos values:
sin(0,0) = 0,000000	cos(0,0) = 1,000000
sin(0,1) = 0,099833	cos(0,1) = 0,995004
sin(0,2) = 0,198669	cos(0,2) = 0,980067
sin(0,3) = 0,295520	cos(0,3) = 0,955336
sin(0,4) = 0,389418	cos(0,4) = 0,921061
sin(0,5) = 0,479426	cos(0,5) = 0,877583
sin(0,6) = 0,564642	cos(0,6) = 0,825336
sin(0,7) = 0,644218	cos(0,7) = 0,764842
sin(0,8) = 0,717356	cos(0,8) = 0,696707

Рисунки 25, 26

Сверху, как уже и писал представлены лишь кусочки работы функций синуса и косинуса.

```
=== Табулированные аналоги ===  
Табулированный Sin:  
x=0,0: original=0,000000, tabulated=0,000000, diff=0,000000  
x=0,1: original=0,099833, tabulated=0,097982, diff=0,001852  
x=0,2: original=0,198669, tabulated=0,195963, diff=0,002706  
x=0,3: original=0,295520, tabulated=0,293945, diff=0,001576  
x=0,4: original=0,389418, tabulated=0,385907, diff=0,003512  
x=0,5: original=0,479426, tabulated=0,472070, diff=0,007355  
x=0,6: original=0,564642, tabulated=0,558234, diff=0,006409  
x=0,7: original=0,644218, tabulated=0,643982, diff=0,000235  
x=0,8: original=0,717356, tabulated=0,707935, diff=0,009421
```

Рисунок 27

Сравнение табулированного синуса и нормальной функции синус, и разница между ними. Как несложно догадаться  $\text{diff} = \text{original} - \text{tabulated}$ .

```
x=0,0: result=1,000000  
x=0,1: result=0,975345  
x=0,2: result=0,970488  
x=0,3: result=0,985429  
x=0,4: result=0,984968  
x=0,5: result=0,970398  
x=0,6: result=0,975624  
x=0,7: result=0,999358  
x=0,8: result=0,975073  
x=0,9: result=0,970586  
x=1,0: result=0,985897  
x=1,1: result=0,984515
```

Рисунок 28

Результаты суммы квадратов синуса и косинуса в зависимости от колва точек.

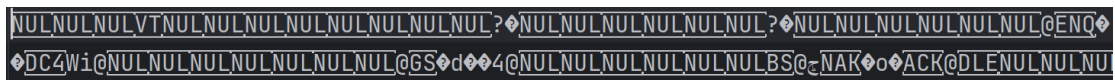


Рисунок 29

Результаты записи табулированной функции экспоненты в бинарном формате в файл exp.txt



Рисунок 30

Результаты записи в файл табулированной функции логарифма. Первое значение - колво точек.

Мои размышления о плюсах и минусах разных способов хранения данных представлены на рисунке 24.

## Пункт №9

Реализации сериализации зависит от того интерфейса который мы имплементируем. Их всего два Serializable и Externalizable. Для реализации Serializable достаточно лишь имплементировать данный интерфейс и создать переменную «private static final long serialVersionUID = 1L;», которая содержит уникальный идентификатор версии сериализованного класса. Больше в данном случае нам делать ничего не надо, потому что то самое имплементирование данного интерфейса служит неким «флагом», говорящим о том, что эту структуру можно сериализовать.

Вот с Externalizable дела обстоят по интереснее. Это «ручной» вариант сериализации. Для неё мы обязательно должны создать конструктор по умолчанию в классе и переопределить 2 метода: writeExternal(ObjectOutput out) и readExternal(ObjectInput in). В моей случае это будет выглядеть вот так (см. Рисунки 31, 32).

```
public void writeExternal(ObjectOutput out) throws IOException {
    // Сериализуем только данные, а не всю структуру массива
    out.writeInt(amountOfElements);

    // Сохраняем координаты X и Y в отдельные массивы
    double[] xValues = new double[amountOfElements];
    double[] yValues = new double[amountOfElements];

    for (int i = 0; i < amountOfElements; i++) {
        xValues[i] = massiveOfPoints[i].getX();
        yValues[i] = massiveOfPoints[i].getY();
    }

    out.writeObject(xValues);
    out.writeObject(yValues);
}
```

Рисунок 31

```

@Override
public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
    // Восстанавливаем данные
    amountOfElements = in.readInt();
    double[] xValues = (double[]) in.readObject();
    double[] yValues = (double[]) in.readObject();

    // Восстанавливаем массив точек
    massiveOfPoints = new FunctionPoint[amountOfElements];
    for (int i = 0; i < amountOfElements; i++) {
        massiveOfPoints[i] = new FunctionPoint(xValues[i], yValues[i]);
    }
}

```

Рисунок 32

Отличие в работе между представленными интерфейсами лучше всего расскажет один из редакторов JavaRush, присваивать чужие знания я себе не буду, поэтому вот:

При использовании `Serializable` под объект просто выделяется память, после чего из потока считываются значения, которыми заполняются все его поля.

Если мы используем `Serializable`, конструктор объекта не вызывается! Вся работа производится через рефлексию (Reflection API, который мы мельком упоминали в прошлой лекции).

В случае с `Externalizable` механизм десериализации будет иным. В начале вызывается конструктор по умолчанию. И только потом у созданного объекта `UserInfo` вызывается метод `readExternal()`, который и отвечает за заполнение полей объекта.

Именно поэтому **любой класс, имплементирующий интерфейс `Externalizable`, обязан иметь конструктор по умолчанию.**

Рисунок 33

Итак, мы разобрались со всем чем только можно в этой лабораторной работе. Что осталось сделать? Правильно! Только проверить корректность работы механизма `Externalizable`, который реализован у меня в работе. Все результаты будут представлены далее.

```

=== Сериализация композиции функций ===
Табулированная функция сериализована в файл: ln_of_exp_serialized.txt
✓ Функция десериализована из файла: ln_of_exp_serialized.txt

Сравнение исходной и десериализованной функции:

```

x	Исходная	Десериализованная	Разница
0,0	0,000000	0,000000	0,000000
1,0	1,000000	1,000000	0,000000
2,0	2,000000	2,000000	0,000000
3,0	3,000000	3,000000	0,000000
4,0	4,000000	4,000000	0,000000
5,0	5,000000	5,000000	0,000000
6,0	6,000000	6,000000	0,000000
7,0	7,000000	7,000000	0,000000
8,0	8,000000	8,000000	0,000000
9,0	9,000000	9,000000	0,000000
10,0	10,000000	10,000000	0,000000

Рисунок 34

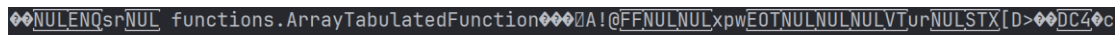


Рисунок 35

А на рисунке 35 представлены данные после процесса сериализации. На сегодня это всё!

Откуда брал информацию:

1. <https://javarush.com/groups/posts/2022-serializacija-i-deserializacija-v-java>
2. <https://javarush.com/groups/posts/2023-znakomstvo-s-interfeysom-externalizable>
3. <https://habr.com/ru/articles/431524/>