ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4

по курсу Объектно-ориентированное программирование

Выполнил: Акст Роман, 6203-010302D

Задание 1

В рамках первого задания были модифицированы классы ArrayTabulatedFunction и LinkedListTabulatedFunction путем добавления конструкторов, принимающих массивы объектов FunctionPoint. Для проверки упорядоченности использовался машинный эпсилон величиной 1e-10, что позволяет корректно сравнивать вещественные числа с учетом погрешности вычислений. На рисунке 1 представлена реализация конструктора с проверкой минимального количества точек и упорядоченности. Особое внимание было уделено обеспечению инкапсуляции - для этого применялось глубокое копирование массива точек через System.arraycopy(), что предотвращает возможность внешнего изменения внутреннего состояния объектов.

```
public ArrayTabulatedFunction(FunctionPoint[] inputPoints)  2 usages  new *
        throws IllegalArgumentException
{
    if(inputPoints.length < 2)
    {
        throw new IllegalArgumentException("Количество точек меньше двух");
    }

    for(int i = 0; i < inputPoints.length - 1; i++)
    {
        if(inputPoints[i].getX() >= inputPoints[i + 1].getX() - EPSILON)
        {
            throw new IllegalArgumentException("Точки не упорядочены по значению абсциссы");
        }
    }

    this.amountOfElements = inputPoints.length;
    this.points = new FunctionPoint[amountOfElements];

    System.arraycopy(inputPoints,  srcPos: 0, this.points,  destPos: 0, amountOfElements);
}
```

Рисунок 1

Задание 2

На следующем этапе работы был создан интерфейс Function, определяющий базовые операции для функций одной переменной. Этот интерфейс включает методы получения границ области определения и вычисления значения функции в заданной точке. Существующий интерфейс TabulatedFunction был переработан таким образом, чтобы наследовать от Function, что логически отражает отношение "табулированная функция является частным случаем функции". На рисунке 2 – интерфейс Function.

```java
package functions;

public interface Function  new *
{
    /**
     * Получает начальную границу интервала определения функции
     * @return координата X левой границы области определения
     */
    double getLeftDomainBorder();  11 implementations  new *

    /**
     * Получает конечную границу интервала определения функции
     * @return координата X правой границы области определения
     */
    double getRightDomainBorder();  11 implementations  new *

    /**
     * Вычисляет приближенное значение функции в указанной точке
     * Использует линейную интерполяцию между соседними точками таблицы
     * @param x координата, для которой вычисляется значение функции
     * @return значение функции в точке x или Double.NaN, если x за границами определения
     */
    double getFunctionValue(double x);  14 implementations  new *

}
```

Рисунок 2

Задание 3

Для реализации аналитически заданных функций был создан пакет functions.basic, содержащий классы элементарных функций. Классы Exp и Log реализуют интерфейс Function непосредственно, в то время как тригонометрические функции организованы через базовый класс TrigonometricFunction. На рисунке 3 – базовый класс TrigonometricFunction. В классе Log был реализован конструктор, принимающий основание логарифма, с соответствующей проверкой допустимости значения. На рисунке 4 – реализация класса Log. Для вычисления значений функций использовались стандартные методы Math.exp(), Math.log(), Math.sin(), Math.cos() и Math.tan(). Области определения были заданы в соответствии с математическими свойствами соответствующих функций.

```java
package functions.basic;

import functions.Function;

public abstract class TrigonometricFunction  3 usages  3 inheritors  new *
    implements Function
{
    @Override  new *
    public double getLeftDomainBorder()
    {
        return Double.NEGATIVE_INFINITY;
    }

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

    @Override  3 implementations  new *
    public abstract double getFunctionValue(double x);
}
```

Рисунок 3

```java
package functions.basic;

import functions.Function;

public class Log  2 usages  new *
    implements Function
{
    double logBase;  2 usages

    public Log(double inputLogBase)  4 usages  new *
    {
        if(inputLogBase < 0)
        {
            throw new IllegalArgumentException("Недопустимое основание логарифма");
        }

        logBase = inputLogBase;
    }

    @Override  new *
    public double getLeftDomainBorder()
    {
        return 0;
    }

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

    @Override  new *
    public double getFunctionValue(double x)
    {
        if(x <= 0)
        {
            return  Double.NaN;
        }

        return Math.log(x) / Math.log(logBase);
    }
}
```

Рисунок 4

Задание 4

Пакет functions.meta содержит классы для комбинирования функций. Были реализованы классы Sum, Mult, Power, Scale, Shift и Composition, каждый из которых представляет определенную операцию над функциями. Особое внимание было уделено корректному определению областей определения результирующих функций. Например, для операций Sum и Mult область определения вычисляется как пересечение областей определения исходных функций, что показано на рисунке 5 - вычисление области определения для Sum. Класс Scale реализует масштабирование по осям координат с учетом возможности отрицательных коэффициентов, а Composition представляет композицию двух функций. Все классы корректно обрабатывают граничные случаи и используют машинный эпсилон для сравнения вещественных чисел.

```
@Override  new *
public double getLeftDomainBorder()
{
    return Math.max(function1.getLeftDomainBorder(), function2.getLeftDomainBorder());
}

@Override  new *
public double getRightDomainBorder()
{
    return Math.min(function1.getRightDomainBorder(), function2.getRightDomainBorder());
}
```

Рисунок 5

Задание 5

Класс Functions был реализован как утилитарный с приватным конструктором, что предотвращает создание его экземпляров. Он содержит статические методы-обертки для создания объектов функций-мет, которые демонстрируют применение принципа фасада - предоставление простого интерфейса для сложной системы классов. На рисунке 6 – реализация класса. Каждый метод делегирует создание соответствующему классу из пакета functions.meta, обеспечивая согласованность и единообразие API.

```
package functions;

import functions.meta.*;

public class Functions  7 usages  new *
{
    private Functions(){}  no usages  new *

    public static Function shift(Function f, double shiftX, double shiftY) { return new Shift(f, shiftX, shiftY); }

    public static Function scale(Function f, double scaleX, double scaleY) { return new Scale(f, scaleX, scaleY); }

    public static Function power(Function f, double power)  4 usages  new *
    {
        return new Power(f, power);
    }

    public static Function sum(Function f1, Function f2) { return new Sum(f1, f2); }

    public static Function mult(Function f1, Function f2) { return new Mult(f1, f2); }

    public static Function composition(Function f1, Function f2) { return new Composition(f1, f2); }
}
```

Рисунок 6

Задание 6

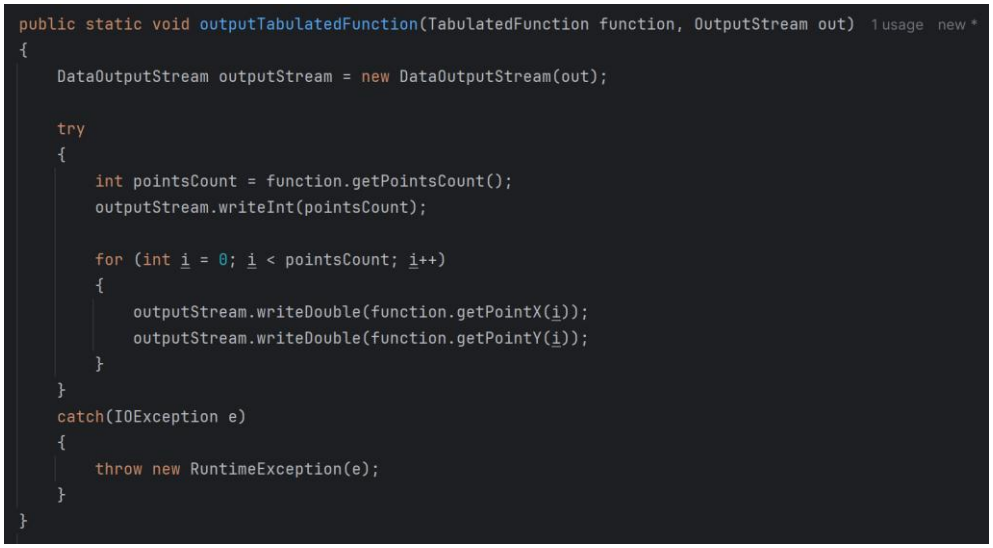
В классе TabulatedFunctions реализован метод tabulate(), который преобразует аналитически заданную функцию в ее табулированное представление. Метод выполняет комплексную проверку входных параметров, включая проверку принадлежности указанного отрезка области определения исходной функции. Для создания точек табуляции используется равномерное разбиение отрезка, при этом значение функции в каждой точке вычисляется с помощью метода getFunctionValue(). На рисунке 7 показана реализация данного метода. В текущей реализации возвращается объект LinkedListTabulatedFunction, что соответствует требованиям задания.

```
public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount)  7 usages  new *
{
    if(function == null)
    {
        throw new IllegalArgumentException("Функция не определена");
    }

    if(leftX >= rightX)
    {
        throw new IllegalArgumentException("Левая граница не может быть больше правой");
    }

    if(pointsCount < 2)
    {
        throw new IllegalArgumentException("Количество точек не может быть меньше двух");
    }

    if(leftX < function.getLeftDomainBorder() - 1e-10 ||
            rightX > function.getRightDomainBorder() + 1e-10)
    {
        throw new IllegalArgumentException("Границы не входят в область определения функции");
    }

    FunctionPoint[] points = new FunctionPoint[pointsCount];

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

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

    return new LinkedListTabulatedFunction(points);
}
```

Рисунок 7

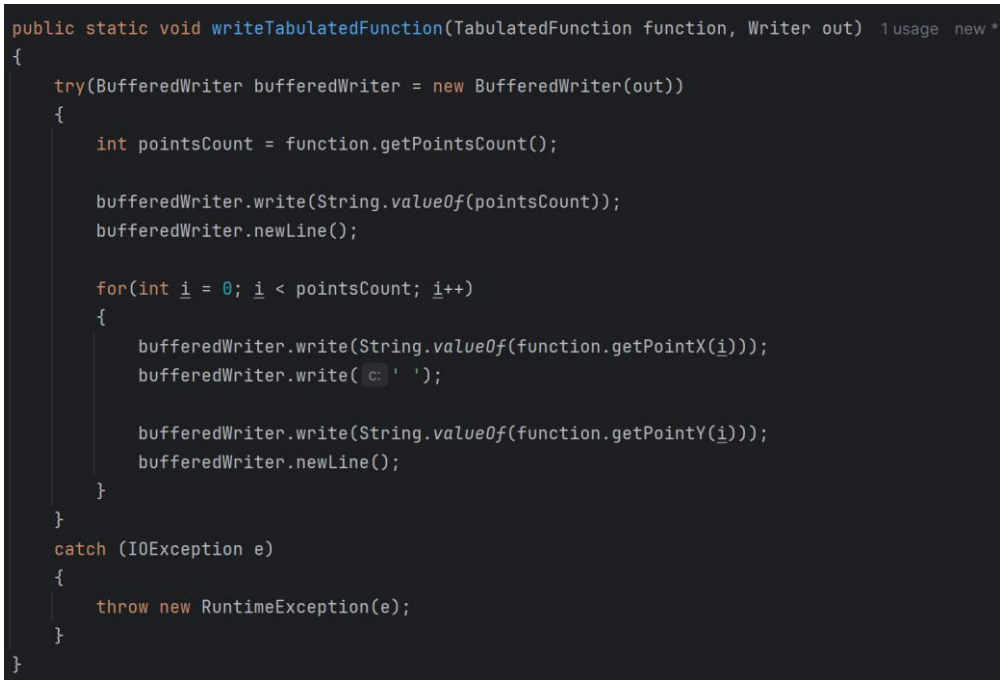
Задание 7

Были реализованы методы ввода-вывода табулированных функций в различных форматах. Для байтовых потоков использовались DataOutputStream и DataInputStream, что обеспечивает компактное представление данных. Для символьных потоков применялись BufferedWriter и StreamTokenizer, что дает читаемый для пользователя формат. На рисунках 8 и 9 – реализации записи в файл с помощью обоих методов. Особенностью реализации является решение не закрывать потоки внутри методов, что оставляет контроль над жизненным циклом потоков вызывающему коду. Исключения IOException преобразуются в RuntimeException для упрощения обработки.



```
public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out)  1 usage  new *
{
    DataOutputStream outputStream = new DataOutputStream(out);

    try
    {
        int pointsCount = function.getPointsCount();
        outputStream.writeInt(pointsCount);

        for (int i = 0; i < pointsCount; i++)
        {
            outputStream.writeDouble(function.getPointX(i));
            outputStream.writeDouble(function.getPointY(i));
        }
    }
    catch(IOException e)
    {
        throw new RuntimeException(e);
    }
}
```

Рисунок 8



```
public static void writeTabulatedFunction(TabulatedFunction function, Writer out)  1 usage  new *
{
    try(BufferedWriter bufferedWriter = new BufferedWriter(out))
    {
        int pointsCount = function.getPointsCount();

        bufferedWriter.write(String.valueOf(pointsCount));
        bufferedWriter.newLine();

        for(int i = 0; i < pointsCount; i++)
        {
            bufferedWriter.write(String.valueOf(function.getPointX(i)));
            bufferedWriter.write( c: ' ');

            bufferedWriter.write(String.valueOf(function.getPointY(i)));
            bufferedWriter.newLine();
        }
    }
    catch (IOException e)
    {
        throw new RuntimeException(e);
    }
}
```

Рисунок 9

Задание 8

Проведено комплексное тестирование реализованной функциональности. Созданы объекты тригонометрических функций, исследована точность их табулированных аналогов при различном количестве точек. Проверено тождество $\sin^2 x + \cos^2 x = 1$ для табулированных функций, что продемонстрировало корректность реализации операций над функциями. На рисунках 10-13 представлены результаты выполнения программы с описанием выполняемых действий. Тестирование работы с файлами подтвердило корректность реализации методов ввода-вывода - функции успешно сохраняются и восстанавливаются без потери точности.

```
Задаем функции синуса и косинуса на отрезке [0;pi] с шагом 0,1:
sin(x):
sin(0,0) = 0,0000
sin(0,1) = 0,0998
sin(0,2) = 0,1987
sin(0,3) = 0,2955
sin(0,4) = 0,3894
sin(0,5) = 0,4794
sin(0,6) = 0,5646
sin(0,7) = 0,6442
sin(0,8) = 0,7174
sin(0,9) = 0,7833
sin(1,0) = 0,8415
sin(1,1) = 0,8912
sin(1,2) = 0,9320
sin(1,3) = 0,9636
sin(1,4) = 0,9854
sin(1,5) = 0,9975
sin(1,6) = 0,9996
sin(1,7) = 0,9917
sin(1,8) = 0,9738
sin(1,9) = 0,9463
sin(2,0) = 0,9093
sin(2,1) = 0,8632
sin(2,2) = 0,8085
sin(2,3) = 0,7457
sin(2,4) = 0,6755
sin(2,5) = 0,5985
sin(2,6) = 0,5155
sin(2,7) = 0,4274
sin(2,8) = 0,3350
sin(2,9) = 0,2392
sin(3,0) = 0,1411
sin(3,1) = 0,0416
```

Рисунок 10

```
cos(x):
cos(0,0) = 1,0000
cos(0,1) = 0,9950
cos(0,2) = 0,9801
cos(0,3) = 0,9553
cos(0,4) = 0,9211
cos(0,5) = 0,8776
cos(0,6) = 0,8253
cos(0,7) = 0,7648
cos(0,8) = 0,6967
cos(0,9) = 0,6216
cos(1,0) = 0,5403
cos(1,1) = 0,4536
cos(1,2) = 0,3624
cos(1,3) = 0,2675
cos(1,4) = 0,1700
cos(1,5) = 0,0707
cos(1,6) = -0,0292
cos(1,7) = -0,1288
cos(1,8) = -0,2272
cos(1,9) = -0,3233
cos(2,0) = -0,4161
cos(2,1) = -0,5048
cos(2,2) = -0,5885
cos(2,3) = -0,6663
cos(2,4) = -0,7374
cos(2,5) = -0,8011
cos(2,6) = -0,8569
cos(2,7) = -0,9041
cos(2,8) = -0,9422
cos(2,9) = -0,9710
cos(3,0) = -0,9900
cos(3,1) = -0,9991
```

Рисунок 11

```
Создаем табулированные аналоги функций и сравниваем с точными:
x = 0,0: sin = 0,0000 (tabulated = 0,0000; error = 0,0000)      cos = 1,0000 (tabulated = 1,0000; error = 0,0000)
x = 0,1: sin = 0,0998 (tabulated = 0,0980; error = 0,0019)      cos = 0,9950 (tabulated = 0,9827; error = 0,0123)
x = 0,2: sin = 0,1987 (tabulated = 0,1960; error = 0,0027)      cos = 0,9801 (tabulated = 0,9654; error = 0,0146)
x = 0,3: sin = 0,2955 (tabulated = 0,2939; error = 0,0016)      cos = 0,9553 (tabulated = 0,9482; error = 0,0072)
x = 0,4: sin = 0,3894 (tabulated = 0,3859; error = 0,0035)      cos = 0,9211 (tabulated = 0,9144; error = 0,0067)
x = 0,5: sin = 0,4794 (tabulated = 0,4721; error = 0,0074)      cos = 0,8776 (tabulated = 0,8646; error = 0,0130)
x = 0,6: sin = 0,5646 (tabulated = 0,5582; error = 0,0064)      cos = 0,8253 (tabulated = 0,8149; error = 0,0105)
x = 0,7: sin = 0,6442 (tabulated = 0,6440; error = 0,0002)      cos = 0,7648 (tabulated = 0,7646; error = 0,0002)
x = 0,8: sin = 0,7174 (tabulated = 0,7079; error = 0,0094)      cos = 0,6967 (tabulated = 0,6884; error = 0,0083)
x = 0,9: sin = 0,7833 (tabulated = 0,7719; error = 0,0114)      cos = 0,6216 (tabulated = 0,6122; error = 0,0094)
x = 1,0: sin = 0,8415 (tabulated = 0,8358; error = 0,0056)      cos = 0,5403 (tabulated = 0,5360; error = 0,0043)
x = 1,1: sin = 0,8912 (tabulated = 0,8840; error = 0,0072)      cos = 0,4536 (tabulated = 0,4506; error = 0,0030)
x = 1,2: sin = 0,9320 (tabulated = 0,9180; error = 0,0140)      cos = 0,3624 (tabulated = 0,3571; error = 0,0052)
x = 1,3: sin = 0,9636 (tabulated = 0,9521; error = 0,0115)      cos = 0,2675 (tabulated = 0,2636; error = 0,0039)
x = 1,4: sin = 0,9854 (tabulated = 0,9848; error = 0,0006)      cos = 0,1700 (tabulated = 0,1699; error = 0,0000)
x = 1,5: sin = 0,9975 (tabulated = 0,9848; error = 0,0127)      cos = 0,0707 (tabulated = 0,0704; error = 0,0003)
x = 1,6: sin = 0,9996 (tabulated = 0,9848; error = 0,0148)      cos = -0,0292 (tabulated = -0,0291; error = -0,0001)
x = 1,7: sin = 0,9917 (tabulated = 0,9848; error = 0,0069)      cos = -0,1288 (tabulated = -0,1285; error = -0,0003)
x = 1,8: sin = 0,9738 (tabulated = 0,9662; error = 0,0076)      cos = -0,2272 (tabulated = -0,2248; error = -0,0024)
x = 1,9: sin = 0,9463 (tabulated = 0,9322; error = 0,0141)      cos = -0,3233 (tabulated = -0,3183; error = -0,0050)
x = 2,0: sin = 0,9093 (tabulated = 0,8981; error = 0,0112)      cos = -0,4161 (tabulated = -0,4117; error = -0,0044)
x = 2,1: sin = 0,8632 (tabulated = 0,8624; error = 0,0008)      cos = -0,5048 (tabulated = -0,5043; error = -0,0006)
x = 2,2: sin = 0,8085 (tabulated = 0,7985; error = 0,0100)      cos = -0,5885 (tabulated = -0,5805; error = -0,0080)
x = 2,3: sin = 0,7457 (tabulated = 0,7345; error = 0,0112)      cos = -0,6663 (tabulated = -0,6567; error = -0,0096)
x = 2,4: sin = 0,6755 (tabulated = 0,6706; error = 0,0049)      cos = -0,7374 (tabulated = -0,7329; error = -0,0045)
x = 2,5: sin = 0,5985 (tabulated = 0,5941; error = 0,0044)      cos = -0,8011 (tabulated = -0,7942; error = -0,0070)
x = 2,6: sin = 0,5155 (tabulated = 0,5079; error = 0,0076)      cos = -0,8569 (tabulated = -0,8439; error = -0,0130)
x = 2,7: sin = 0,4274 (tabulated = 0,4217; error = 0,0056)      cos = -0,9041 (tabulated = -0,8937; error = -0,0104)
x = 2,8: sin = 0,3350 (tabulated = 0,3347; error = 0,0003)      cos = -0,9422 (tabulated = -0,9410; error = -0,0012)
x = 2,9: sin = 0,2392 (tabulated = 0,2367; error = 0,0025)      cos = -0,9710 (tabulated = -0,9583; error = -0,0127)
x = 3,0: sin = 0,1411 (tabulated = 0,1387; error = 0,0024)      cos = -0,9900 (tabulated = -0,9755; error = -0,0145)
x = 3,1: sin = 0,0416 (tabulated = 0,0408; error = 0,0008)      cos = -0,9991 (tabulated = -0,9928; error = -0,0063)
```

Рисунок 12

```
Сумма квадратов табулированных синуса и косинуса:
x = 0,0: sin² + cos² = 1,0000
x = 0,1: sin² + cos² = 0,9753
x = 0,2: sin² + cos² = 0,9705
x = 0,3: sin² + cos² = 0,9854
x = 0,4: sin² + cos² = 0,9850
x = 0,5: sin² + cos² = 0,9704
x = 0,6: sin² + cos² = 0,9756
x = 0,7: sin² + cos² = 0,9994
x = 0,8: sin² + cos² = 0,9751
x = 0,9: sin² + cos² = 0,9706
x = 1,0: sin² + cos² = 0,9859
x = 1,1: sin² + cos² = 0,9845
x = 1,2: sin² + cos² = 0,9703
x = 1,3: sin² + cos² = 0,9759
x = 1,4: sin² + cos² = 0,9987
x = 1,5: sin² + cos² = 0,9748
x = 1,6: sin² + cos² = 0,9707
x = 1,7: sin² + cos² = 0,9864
x = 1,8: sin² + cos² = 0,9841
x = 1,9: sin² + cos² = 0,9702
x = 2,0: sin² + cos² = 0,9762
x = 2,1: sin² + cos² = 0,9981
x = 2,2: sin² + cos² = 0,9745
x = 2,3: sin² + cos² = 0,9708
x = 2,4: sin² + cos² = 0,9869
x = 2,5: sin² + cos² = 0,9836
x = 2,6: sin² + cos² = 0,9702
x = 2,7: sin² + cos² = 0,9765
x = 2,8: sin² + cos² = 0,9975
x = 2,9: sin² + cos² = 0,9743
x = 3,0: sin² + cos² = 0,9709
x = 3,1: sin² + cos² = 0,9873
```

Рисунок 13

```
При 5 точках:
x = 0,0: значение = 1,000000
x = 0,5: значение = 0,864487
x = 1,0: значение = 0,883675
x = 1,5: значение = 0,951957
x = 2,0: значение = 0,854819
x = 2,5: значение = 0,912382
x = 3,0: значение = 0,913432

При 10 точках:
x = 0,0: значение = 1,000000
x = 0,5: значение = 0,970398
x = 1,0: значение = 0,985897
x = 1,5: значение = 0,974808
x = 2,0: значение = 0,976203
x = 2,5: значение = 0,983628
x = 3,0: значение = 0,970920

При 20 точках:
x = 0,0: значение = 1,000000
x = 0,5: значение = 0,999363
x = 1,0: значение = 0,998756
x = 1,5: значение = 0,998181
x = 2,0: значение = 0,997638
x = 2,5: значение = 0,997125
x = 3,0: значение = 0,996644

При 50 точках:
x = 0,0: значение = 1,000000
x = 0,5: значение = 0,999339
x = 1,0: значение = 0,999012
x = 1,5: значение = 0,999017
x = 2,0: значение = 0,999357
x = 2,5: значение = 0,999971
x = 3,0: значение = 0,999322
```

Рисунок 14

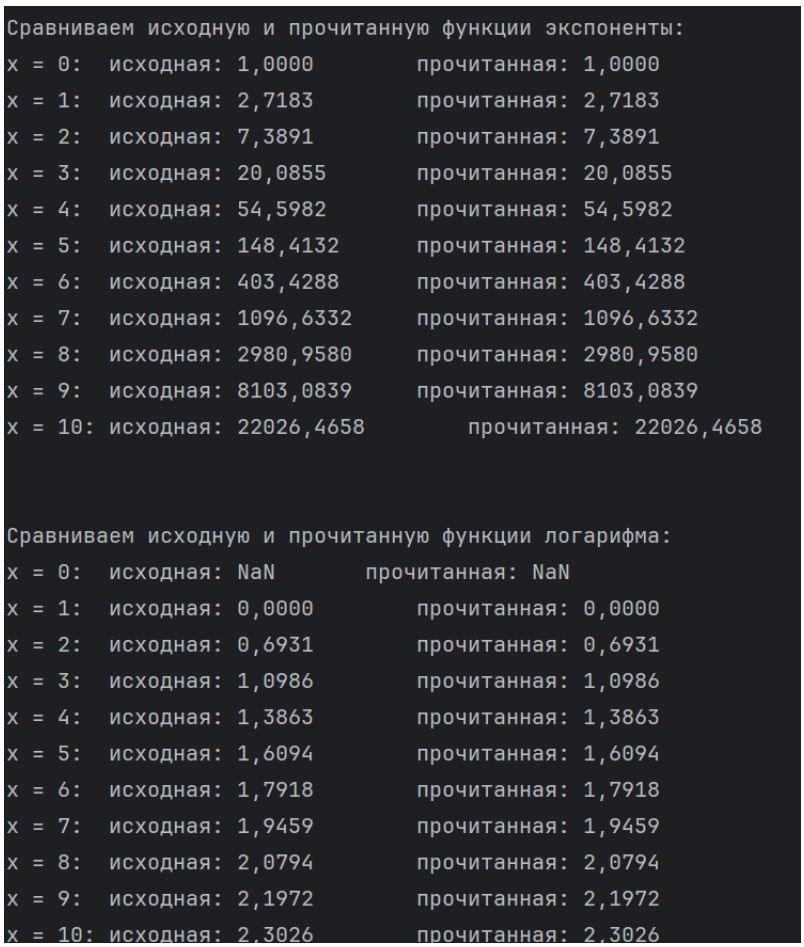

Сравниваем исходную и прочитанную функции экспоненты:
x = 0: исходная: 1,0000 прочитанная: 1,0000
x = 1: исходная: 2,7183 прочитанная: 2,7183
x = 2: исходная: 7,3891 прочитанная: 7,3891
x = 3: исходная: 20,0855 прочитанная: 20,0855
x = 4: исходная: 54,5982 прочитанная: 54,5982
x = 5: исходная: 148,4132 прочитанная: 148,4132
x = 6: исходная: 403,4288 прочитанная: 403,4288
x = 7: исходная: 1096,6332 прочитанная: 1096,6332
x = 8: исходная: 2980,9580 прочитанная: 2980,9580
x = 9: исходная: 8103,0839 прочитанная: 8103,0839
x = 10: исходная: 22026,4658 прочитанная: 22026,4658
Сравниваем исходную и прочитанную функции логарифма:
x = 0: исходная: NaN прочитанная: NaN
x = 1: исходная: 0,0000 прочитанная: 0,0000
x = 2: исходная: 0,6931 прочитанная: 0,6931
x = 3: исходная: 1,0986 прочитанная: 1,0986
x = 4: исходная: 1,3863 прочитанная: 1,3863
x = 5: исходная: 1,6094 прочитанная: 1,6094
x = 6: исходная: 1,7918 прочитанная: 1,7918
x = 7: исходная: 1,9459 прочитанная: 1,9459
x = 8: исходная: 2,0794 прочитанная: 2,0794
x = 9: исходная: 2,1972 прочитанная: 2,1972
x = 10: исходная: 2,3026 прочитанная: 2,3026

Рисунок 15

Задание 9

Реализована сериализация объектов табулированных функций с использованием интерфейса Externalizable. Этот подход был выбран за возможность тонкого контроля над процессом сериализации. В классах ArrayTabulatedFunction и LinkedListTabulatedFunction реализованы методы writeExternal() и readExternal(), которые обеспечивают сохранение и восстановление состояния объектов. На рисунке 16 показаны результаты тестирования. Тестирование подтвердило корректность работы механизма сериализации - десериализованные объекты полностью идентичны исходным(при x = 0 функция не определена, из-за чего нельзя сказать об идентичности). Преимуществом Externalizable является компактность результирующих файлов и явный контроль над версионированием формата данных.

```
Табулированная функция сериализована в файл
Функция десериализована из файла

Сравнение исходной и десериализованной функции:
x       Исходная    Десериализованная   Совпадают
0,0     NaN NaN     false
1,0     1,000000    1,000000        true
2,0     2,000000    2,000000        true
3,0     3,000000    3,000000        true
4,0     4,000000    4,000000        true
5,0     5,000000    5,000000        true
6,0     6,000000    6,000000        true
7,0     7,000000    7,000000        true
8,0     8,000000    8,000000        true
9,0     9,000000    9,000000        true
10,0        10,000000   10,000000       true
```

Рисунок 16